A
Comparative Analysis of
ASCII and XML Logging Systems

THESIS

Eric Hanington, Civilian, USAF

AFIT/GCO/ENG/10-17

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/GCO/ENG/10-17

A
Comparative Analysis of
ASCII and XML Logging Systems

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Eric Hanington, Bachelor of Science in Computer Science

Civilian, USAF

Sept 2010

AFIT/GCO/ENG/10-17

A
Comparative Analysis of
ASCII and XML Logging Systems

Eric Hanington, Bachelor of Science in Computer Science
Civilian, USAF

Approved:

/signed/	10 Sept 2010
Dr. Rusty Baldwin, (Chairman)	date
/signed/	10 Sept 2010
Dr. Michael Grimaila (Member)	date
/signed/	10 Sept 2010
Dr. Robert Mills (Member)	date

AFIT/GCO/ENG/10-17

Abstract

This research compares XML and ASCII based event logging systems in terms of their storage and processing efficiency. XML has been an emerging technology, even for security. Therefore, it is researched as a logging system with the mitigation of its verbosity. Each system consists of source content, the network transmission, database storage, and querying, which are all studied as individual parts. The ASCII logging system consists of the text file as source, FTP as transport, and a relational database system for storage and querying. The XML system has the XML files and XML files in binary form using Efficient XML Interchange encoding, FTP as transport using both XML and binary XML, and an XML database for storage and querying. Further comparisons are made between the XML itself and binary XML, as well as binary XML to ASCII text when comparing file sizes such as, ASCII = 2211.58 KB, XML = 3463.56 KB, and binary XML = 425 KB. As well as, transmission efficiency having times such as, ASCII = 12 msec, XML = 26 msec, and binary XML = 4 msec. XML itself is a poor choice compared to ASCII for hard drive being 1.6 times greater and network transport time being 2.2 times longer. However, in a binary form compared to ASCII, it uses less hard drive space, that is 0.19 times the size, and network resources being a 1/3. Because no XML databases support a binary XML, it is loaded without any optimization. The ASCII loads into the relational database with less time than XML into its database. ASCII's loading time could load in 1.9 seconds compared to the XML loading in 7.7 sec. However, querying each database, neither outperforms the other as one query results in shorter time for one, and another query results in a shorter time for the other. One query resulted in the relational database executing in 3 sec and the XML database in 85 sec. Where as, another query has relational database taking 37 sec and the XML database 6 sec. Therefore, XML and/or its binary form is a viable candidate for use as a comprehensive logging system.

Acknowledgements

First and foremost, I owe a large debt of gratitude to my family for their encouragement and support. I also am in much debt to my advisor for his countless times in guiding me in the writing of my thesis. As well as my other committee members in their resources and advise given. Finally, for staff at the Air Force Institute of Technology for the resources that they let me use for my thesis.

Eric Hanington

Table of Contents

Page

List of Figures

List of Tables

A

Comparative Analysis of ASCII and XML Logging Systems

I. An XML logging system?

Since the creation of Extensible Markup Language (XML) by the World Wide Web Consortium in the late 1990's there has been initiatives to create XML versions of audit logs [HB99] [XLF01]. The advantage of XML logs is that they are in a standard format for processing. This in a sense is true. However, at the time, and to a lesser extent today, XML is much larger than the simpler ASCII text files. The question then arises as to how to read, search, and analyze the XML.

Much has changed since the time that XML was initially created. Many more standards and technologies have been built on top of the core of XML. A continuing issue, however, is data naming collisions. In examining multiple logs, there is more than one labeled "user". What does "user" mean in each context? One solution is to give each labeling a scope such as linuxAccount:user, relationalDatabaseAccount:user, which provides context. XML has such a mechanism: XML Namespaces.

Another XML feature is the XML Schema. This is a separate document that specifies what data is permitted into the document and how the document is laid out. This provides a filter for audit or similar data to make sure that it conforms to expected standards and formats.

However, it still remains that the XML data must be searched. Thus in mid-2000s, a querying language was designed specifically for XML [Wal07]. XML can now be placed into a database of its own for searching. The meaning / context of a parcel of information is known since XML's "markup" is self-documenting.

Finally, the newest technology of significance for this effort is Efficient XML Interchange [Wor08]. This new standard converts XML from its well-documented

verbose form, into binary form which is compact and designed for efficient processing on the receiving end.

This research is to determine if an XML logging system is feasible in comparison to an ASCII logging system. XML has been an emerging technology, even for security such as Security Content Automation Protocol (SCAP) [Cor08]. Therefore, integrating an XML logging system with other XML-based security makes security tools and technology easier to build and use. XML logging information also becomes extensible, resulting in easier future adaptations. Both of these benefits can save time, money, and other resources.

Since there is limited time and resources, a full-scale implementation of a network and other resources is not done. Instead, the technologies discussed above are used to form "island" processes that could form the basis of a comprehensive logging system. These islands focus on hard drive usage, network transmission time, coding and decoding times, and database times. The times recorded for network traffic are the least realistic since all communications are connected from one computer to another via a switch. The databases' times are also not as realistic, since they are not set up on real-world servers.

More details are given in their appropriate chapters.

II. Background Material

There is the need for recording activities and keeping other records of certain activities. Financial system logs, for example, log capital inflow and expenditures outflow. For inventory applications there are shipping logs that may include identification information, time arrived, and the origin of the item; or for items to be shipped, item identification, destination, and departure time. In the medical field, very detailed records are kept including who saw whom, and what care the provider gave to the patient. It is no different for computers and networks. However, the number and volume of the records or logs can be quite high. Logs are each tailored to a particular service, product, or standard. Often, multiple logs may need to be correlated. XML and the related XQuery standard can aide in this task.

2.1 Extensible Markup Language

Extensible Markup Language (XML) has been used to transport logging information [LOG01], [CCC04], [GPR+03]. However, additional technologies and standards have made doing so more practical. Since XML is by design extensible, it is easy to adapt to current standards and technologies, as well as to new standards and technologies. The markup capability of XML can separate the data itself from the presentation of data, as is shown in Figure 2.1. The data $favorite$ is marked up to be presented differently than its accompanying data using elements (element-pair) $< i >$ and $< /i >$. The "language" attribute also demonstrates the separation of data and presentation, as the "name" is tailored to the language of the audience. Since logs are not typically designed with presentation in mind, XML provides a means to do so for data. This enables the information-system as well as end-users to process the logged data. Another feature of XML is its hierarchical layout in which child-elements can be repeated. These features are discussed further in the querying logs section.

```
<catalog>
    <product dept="WMN">
        <number>557</number>
        <name language="en">Fleece Pullover</name>
        <colorChoices>navy black</colorChoices>
    </product>
    <product dept="ACC">
        <number>563</number>
        <name language="en">Floppy Sun Hat</name>
    </product>
    <product dept="ACC">
        <number>443</number>
        <name language="en">Deluxe Travel Bag</name>
    </product>
    <product dept="MEN">
        <number>784</number>
        <name language="en">Cotton Dress Shirt</name>
        <colorChoices>white gray</colorChoices>
        <desc>Our <i>favorite</i> shirt! </desc>
    </product>
</catalog>
```

Figure 2.1: Example XML: "Catalog"

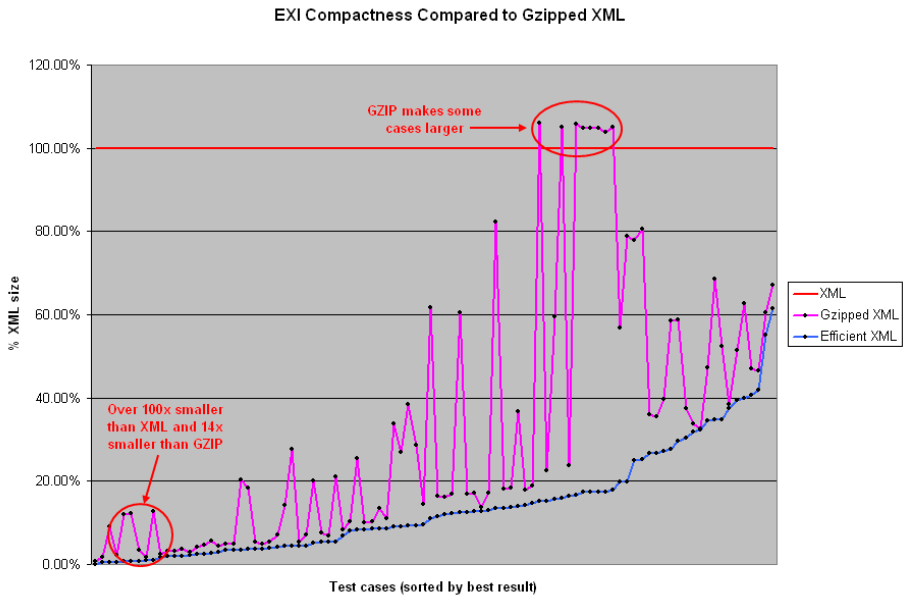

Figure 2.2: EXI Compactness compared to Gzipped XML [EXI09] Test cases sorted by best result

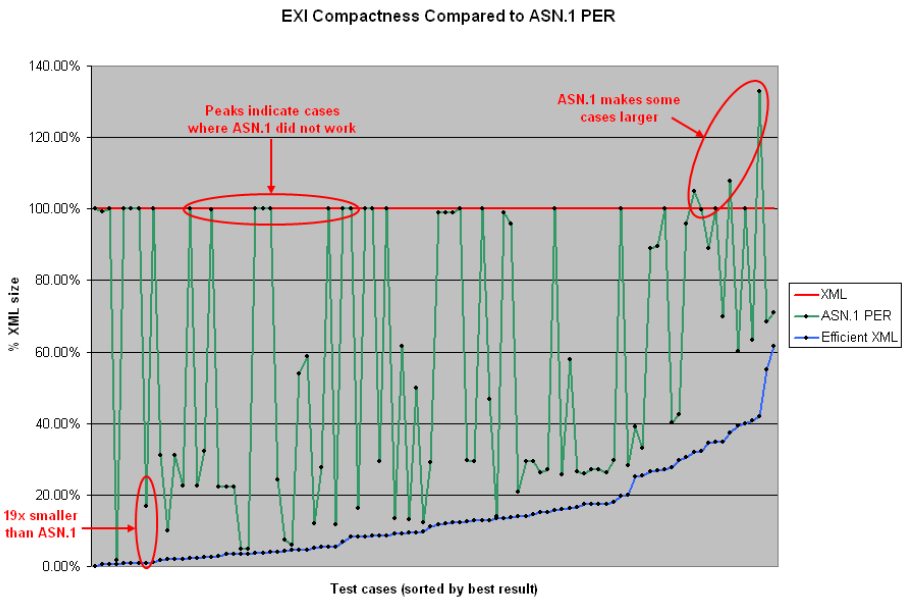

Figure 2.3: EXI Compactness compared to Fast Infoset [EXI09] Test cases sorted by best result

XML's verbosity is a significant drawback. Although it makes the document as a whole and the data elements portable, self-explanatory, and easier to design for various processing methods, this results in a document that may be significantly larger, compared to the data itself. This was considered by the XML designers, the World Wide Web Consortium (W3C), and resulted in the formation of the XML Binary Characterization Working Group. [Wor03]. The Binary Characterization Working Group determined a Binary XML was warranted based on case studies [WG05]. Binary XML reduces disk, memory, and network bandwidth requirements, as well as improves processing performance.

The Efficient XML Interchange (EXI) Working Group [Wor08] is to create an open royalty-free, Binary XML standard. The "EXI" binary format has been tested against other binary implementations of XML such as XML+*gzip*, as shown in Figure 2.2, and Fast Infoset, as shown in Figure 2.3. Fast Infoset is defined using the Abstract Syntax Notation 1 (ASN.1) which specifies the components of binary data. It is then encoded using the Packed Encoding Rule (PER), as is referred to in Figure 2.3. EXI

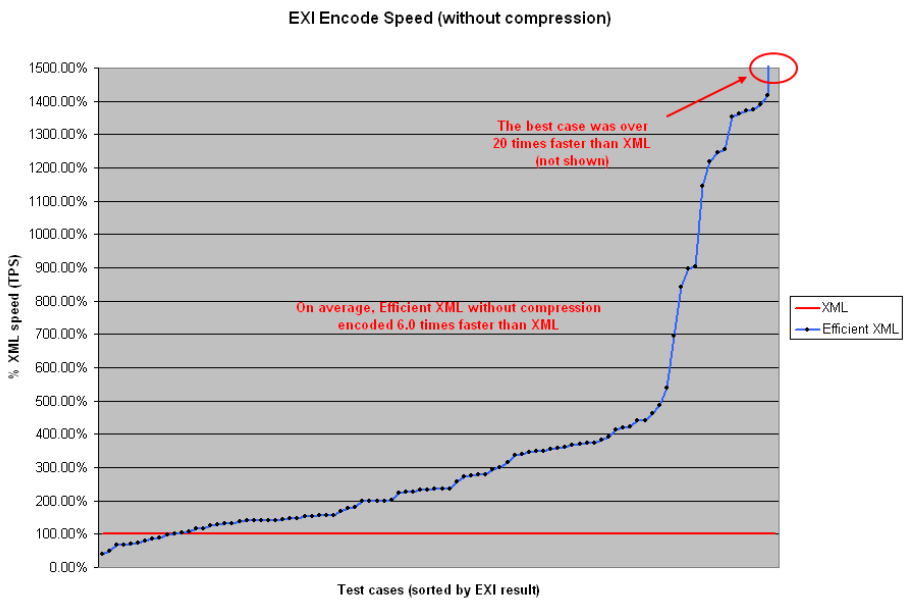

Figure 2.4: EXI Encoding Speed without compression [EXI09] Test cases sorted by EXI result

encodes and decodes faster, based on TPS as defined by the previous working group and as shown in figures 2.4, 2.5 respectively. The compacting and processing speed is achieved by stripping out the information or performing a Huffman-like codec for all information known to have a finite set of values. These values, known in advance, are provided by an XML Schema. XML Schema, shown in Figure 2.6, is an XML standard that gives XML structure by specifying constraints on the layout of the XML document and data it contains within it. This is shown in the figure, demonstrating the specifying of the hierarchy for "catalog" to have *number*, *name*, *colorchoices*, and *desc* be children of the element *product*; *product* is to have an attribute "dept". The figure also demonstrates the restricting of a value set; the "language" attribute will only use "en" and "fr".

2.2 Logs

Logs or audit trails are an important component of system security analysis that can verify conformance with security policies and trace the cause of policy

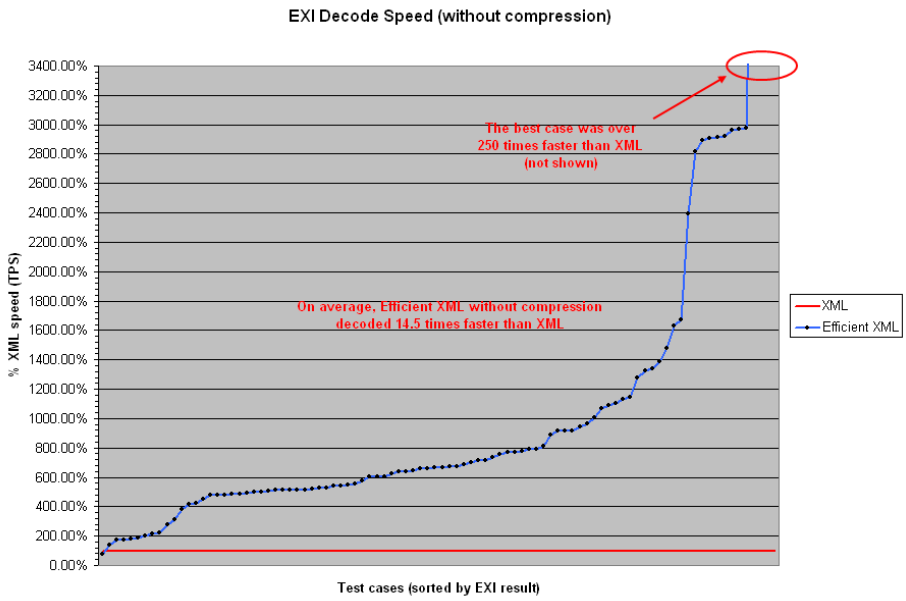

Figure 2.5: EXI Decode Speed without compression [EXI09] Test cases sorted by EXI result

breaches. Logs are typically one of three types: Binary, ASCII, or Formatted/Marked-up (XML).

Binary type logs have the advantage of being the fastest and the most compact. However, they are not human-readable as is shown in Figure 2.7, nor easily portable. Thus, the use of specific tools to view logs, as shown in Figure 2.8 and to search them is required. Although the binary type logs are not human-readable, this can mitigate an attacker's attempt to hide a break-in as some tools attempt to purge event entries from binary logs which could corrupt them.

ASCII logs are very common on UNIX systems and are easy to use and understand, but are not as compact as binary logs. However, they are human and machine readable and very portable, as is shown in Figure 2.9. ASCII logs come in a variety of different formats, i.e., comma-delimited rows, tab-delimited rows, key/value pairs, other similar format, or strings of free-text.

XML logs are not as common as the other two types of logs. An XML log is technically a text-based log, but is treated differently since XML is structured and

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="catalog" type="CatalogType"/>
  <xs:complexType name="CatalogType">
    <xs:sequence>
      <xs:element ref="product" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="product" type="ProductType"/>
  <xs:complexType name="ProductType">
    <xs:sequence>
      <xs:element name="number" type="xs:integer"/>
      <xs:element name="name" type="NameType"/>
      <xs:element name="colorChoices" type="ColorListType" minOccurs="0"/>
      <xs:element name="desc" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="dept" type="xs:string"/>
   </xs:complexType>
  <xs:simpleType name="ColorListType">
    <xs:list itemType="xs:string"/>
  </xs:simpleType>
  <xs:complexType name="NameType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="language" type="LangType"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:simpleType name="LangType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="en"/>
      <xs:enumeration value="fr"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Figure 2.6: Example XML Schema: "Catalog"

uses UTF-8 or UTF-16 character encoding, as opposed to ASCII. Binary XML is also considered "XML" rather than a binary log since it retains its XML properties. Despite the popularity of XML as a medium of communication, it is not as "user-friendly" for a person to read directly, but can be done. It is uncommon as a logging format.

2.3 XML Logs

There are standards for XML logs. However, XML logs are often specified and implemented for a particular technology or organization, such as the "XML Log Standards of Digital Libraries" [GPR+03]. XML for data in transport is more common. In intrusion detection systems, standards such as the "Intrusion Detection Message Exchange Format" (IDMEF) [IDM04] and Security Event Exchange (SDEE) [SDE]

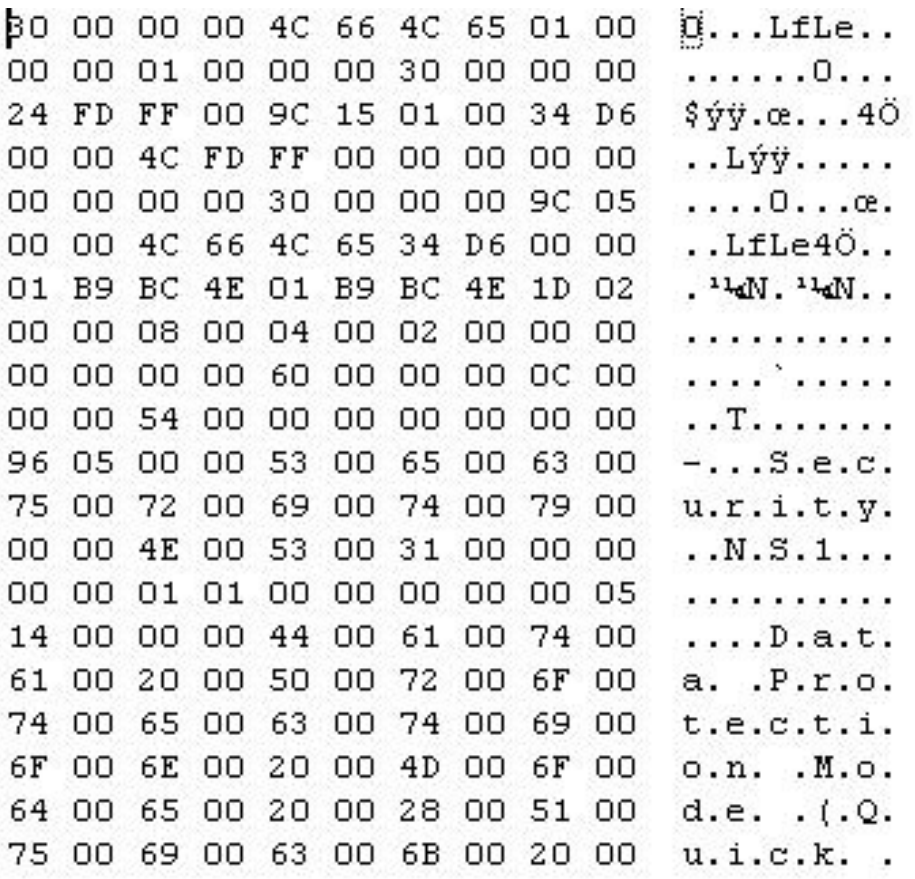

Figure 2.7: Binary Log shown in hexadecimal ASCII strings

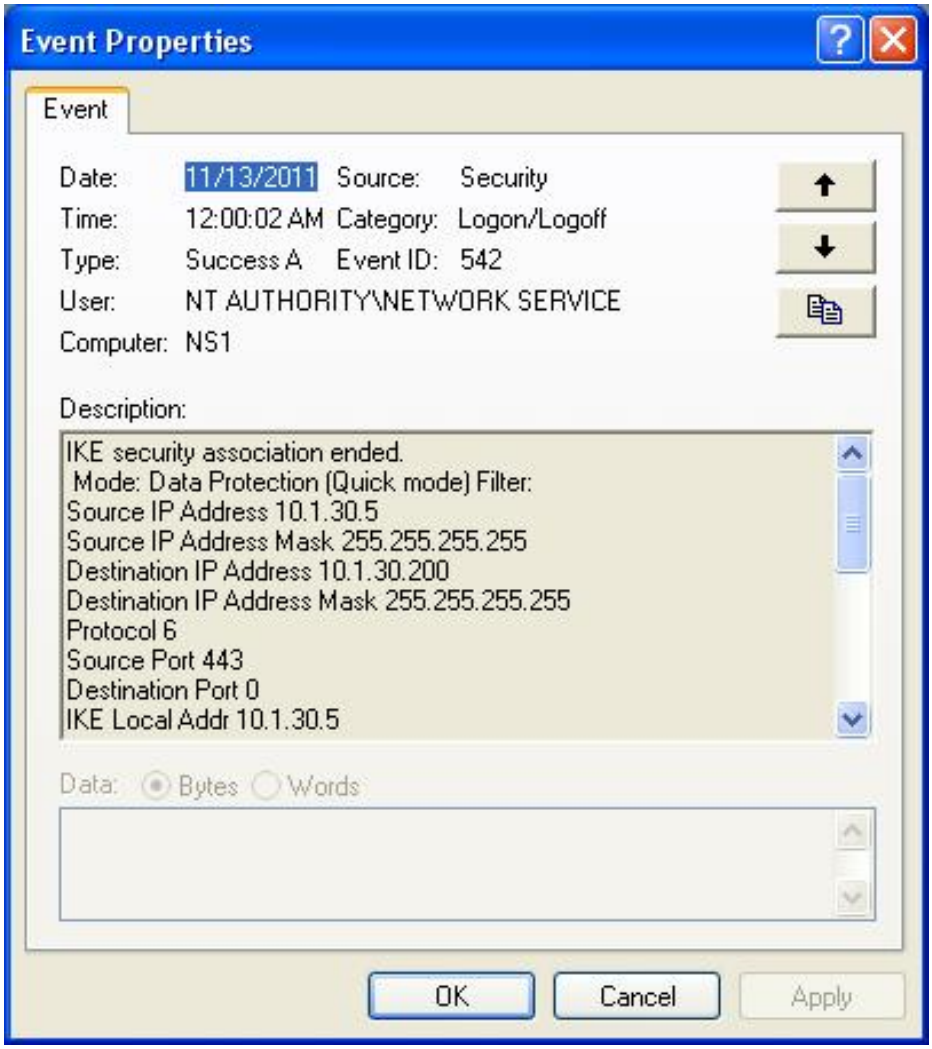

Figure 2.8: Binary Log Viewer

```
127.0.0.1 - - [10/Aug/2006:20:28:10 -0400] "GET / HTTP/1.1" 200 44
127.0.0.1 - - [10/Aug/2006:20:47:54 -0400] "GET / HTTP/1.1" 200 44
127.0.0.1 - - [10/Aug/2006:20:47:55 -0400] "GET /favicon.ico HTTP/1.1" 404 209
127.0.0.1 - - [10/Aug/2006:20:47:55 -0400] "GET /favicon.ico HTTP/1.1" 404 209
127.0.0.1 - - [10/Aug/2006:20:50:14 -0400] "GET / HTTP/1.1" 200 44
192.168.7.115 - - [10/Aug/2006:21:05:03 -0400] "GET / HTTP/1.1" 200 44
192.168.7.115 - - [10/Aug/2006:21:05:04 -0400] "GET /favicon.ico HTTP/1.1" 404 209
192.168.7.115 - - [10/Aug/2006:21:05:04 -0400] "GET /favicon.ico HTTP/1.1" 404 209
192.168.7.115 - - [10/Aug/2006:21:06:08 -0400] "GET /favicon.ico HTTP/1.1" 404 209
192.168.7.115 - - [10/Aug/2006:21:06:15 -0400] "GET /favicon.ico HTTP/1.1" 404 209
192.168.7.115 - - [10/Aug/2006:21:06:21 -0400] "GET // HTTP/1.1" 200 44
192.168.7.115 - - [10/Aug/2006:21:06:32 -0400] "GET /%5C HTTP/1.1" 404 199
192.168.7.115 - - [10/Aug/2006:21:06:41 -0400] "GET /c:%5C HTTP/1.1" 404 201
192.168.7.115 - - [10/Aug/2006:21:06:49 -0400] "GET /%5C/ HTTP/1.1" 404 200
192.168.7.115 - - [11/Aug/2006:23:34:20 -0400] "GET / HTTP/1.1" 200 44
192.168.7.115 - - [11/Aug/2006:23:34:23 -0400] "GET /favicon.ico HTTP/1.1" 404 209
192.168.7.115 - - [11/Aug/2006:23:34:24 -0400] "GET /favicon.ico HTTP/1.1" 404 209
192.168.7.115 - - [12/Aug/2006:02:31:30 -0400] "GET / HTTP/1.1" 304 -
192.168.7.115 - - [12/Aug/2006:02:31:51 -0400] "GET /download HTTP/1.1" 404 206
```

Figure 2.9: ASCII Log

Figure 2.10: SEAL Revision 1

use XML as a means of transport. DEViSE is an open middleware architecture that manages and coordinates information between security visualization tools using XML [RXB09].

XML logs are sometimes proprietary as is Microsoft's Windows Vista [Sch07]. XML logs fixed problems with the old Windows NT event logging system which loaded large tables into memory and was unable to query or filter log entries. Although an XML log can be verbose, it was used in the low-cost compact Spatial Environmental Autonomous Logger (SEAL) shown in Figures 2.10 and 2.11. SEAL can be used in non-real-time unmanned aerial vehicles. [CC08]

There have been efforts to create a universal log system. The Extensible Log Format Initiative created a universal format based on XML [HB99]. The LOGML format was last used in the proof-of-concept "web usage mining" in 2002 [PKZ02].

IBM created a universal log system using their Common Base Event (CBE) standard in the Common Event Infrastructure (CEI), their implementation of the

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<gpx creator="CSOIS SEAL Datalogger" version="1.1">
<trk>
  <trkpt lat="41.74318700" lon="-111.80720000">
  <gpslock>1</gpslock>
  <time>2008-02-04T03:22:56.164Z</time>
      <ele>1361.798</ele>
      <speed>0.001</speed>
      <co2>132</co2>
      <accel_y>2</accel_y>
      <temperature>11</temperature>
      <nh3>502</nh3>
  </trkpt>
  <trkpt>
  ....
  </trkpt>
</trk>
</gpx>
```

Figure 2.11: SEAL's XML log

"Web Service Distribution Management" (WSDM) standard from the Organization for the Advancement of Structured Information Standards (OASIS). The CBE specifies how events are to be formatted [Cor03] [IBM03], while storage and processing of the XML documents is left to applications. To simplify implementation and promote the CBE, a set of APIs known collectively as CEI was developed. These APIs create, distribute, and process CBEs which are configured by the user, as dictated by business and performance needs.

Another universal logging system is MITRE 's Common Event Expression (CEE) standard [CEE08]. CEE has four components: Taxonomy, Syntax, Transport, and Recommendations, as displayed in Figure 2.12. The categorizing, "labeling" of syntax, and choosing the most appropriate transport method makes the standard much more abstract and adaptable, which could lead to wider adoption of this standard. CEE members are still determining how to build the standard from use-cases and mailing-list discussions.

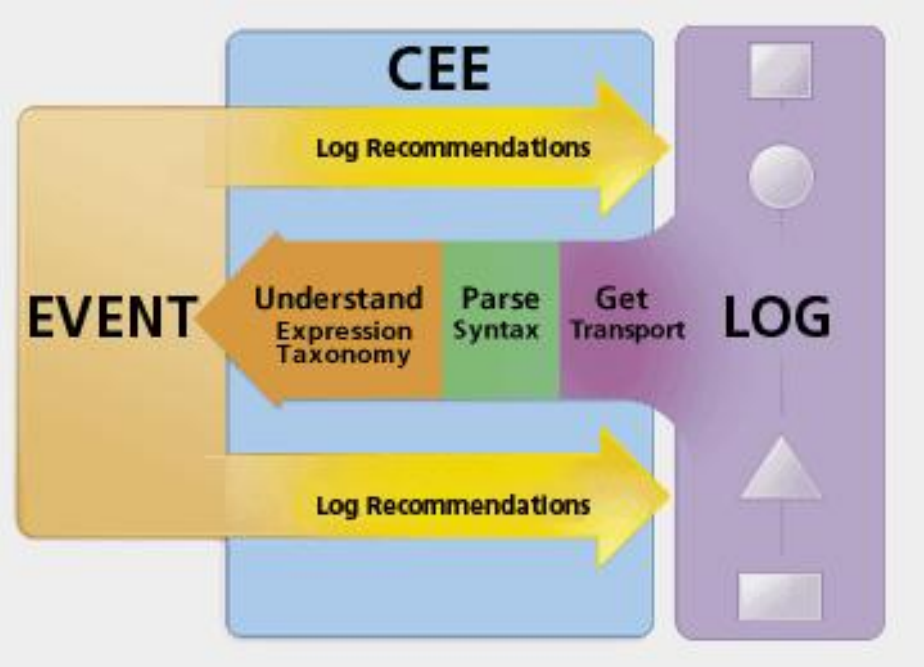

Figure 2.12: Common Event Expression chart

2.4 Querying Logs

Querying logs directly is problematic. It is not possible to query a log of binary data without a tool designed specifically for that purpose. Even then, one is limited to the capabilities of the tool itself. Basic ASCII-text files can be queried to some extent using scripting tools such as PERL. The difficulty is designing combinations of retrievable data or developing multiple scripts for the individual combinations. Often, log data is transferred or copied into a database management system (DBMS), frequently a relational database (RDBMS). The RDBMS works well for logs that store line-entries for events or for tuple-like data. RDBMSs do not handle log data that have components with subcomponent(s) or a list of data which must be decoupled. XML databases, on the other hand, process component-subcomponent and component-list data organization very well. In the query shown in Figure 2.13, the query statement returns every third "product" element and associated children from the "catalog" [Wal07].

```
doc("catalog.xml")/catalog/product[position() mod 3 = 0]
```

Figure 2.13: Example: simple XQuery query statement

2.5 Summary

Because of its self-documenting ability and hierarchical structure, XML facilitates the analysis of log data. Because it is text it is also portable. Additional XML standards make querying an XML log simpler than binary or ASCII logs queries. Although XML logs are "specialized" when used in particular open standards or proprietary products, they lend themselves to standards that complements XML's structure.

III. Methodology

3.1 Introduction

XML is ideal for compartmentalizing information, is ubiquitous, and is verbose. The first two characteristics of XML, its ability to compartmentalize information and its ubiquitousness, suggest it may be useful for logging audit data. However, XML's verboseness poses a challenge, particularly in systems that generate a significant amount of data. Therefore, the question arises: Are XML computer logs a reasonable use of this technology?

3.2 Problem Definition

3.2.1 Goals and Hypothesis. The goal of this research is to determine if XML logs use fewer resources and less bandwidth to transport after being encoded into binary XML (which preserves XML characteristics) than ASCII logs. A relational/object-relational database containing ASCII log data and an XML database with XML log data are compared.

It is known that XML logs do in fact use more computer and network resources than ASCII logs. However, XML logs encoded with or without an XML Schema might use fewer resources than ASCII logs. It is expected that binary XML with a schema will use fewer resources than one without a schema.

Furthermore, it is expected that an XML database with a schema will outperform other audit databases. Without a schema, the XML database can only presume the data is "generic" and cannot be optimized. A relational/object-relational database, however, is configured for optimization, and will likely outperform an XML database without a schema.

3.2.2 Approach. This research compares an ASCII Audit System to an XML Audit System. The ASCII system has one form, while the XML system has two. As part of the comparison, ASCII will be compared to both XML forms. In addition,

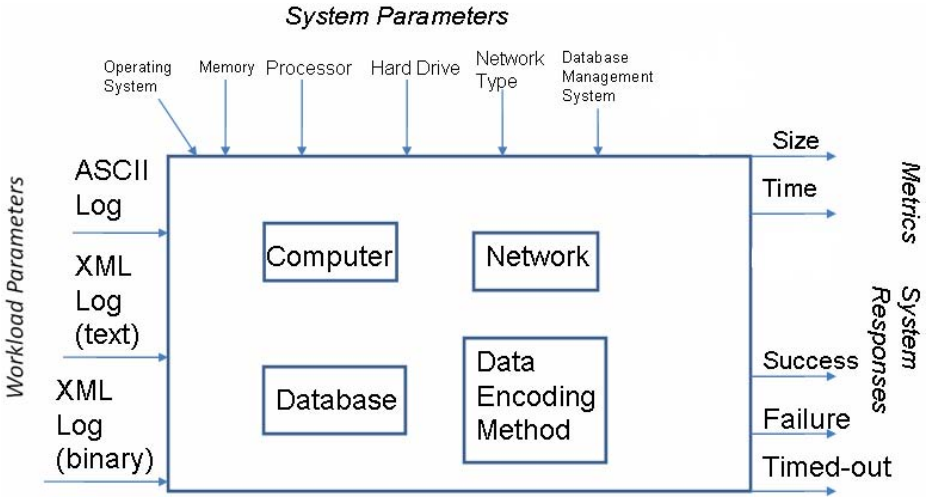

Figure 3.1: System Under Test

the two XML forms are compared to each other. Thus, which audit system is better overall is determined.

3.3 System Under Test

The System Under Test (SUT) is the Audit Logging System, as shown in Figure 3.1. The SUT has four components: a computer, a network, a database, and the data encoding method. The SUT does not measure the execution time of the software nor the hardware system that generates audit events. Therefore, how fast and how many resources are used to create ASCII versus XML versus XML binary audit data is not addressed. Rather, the computer's storage, and network's data transmission resources are measured, based on the submitted workload.

3.4 System Services

The SUT provides the services of audit data storage, audit data transmission, and query of audit data. The Computer provides the storage service, with the outcomes of success or failure. The storage service is successful if audit data can be read and written to the hard drive, and is a failure otherwise.

The audit data transmission service is provided by the Network. It has service outcomes of success or timed-out. The service is a success if all audit data to be sent arrives without data loss or corruption. If a time constraint is associated with the test undertaken, then the data sent must also be within the time constraint. Finally, if the data is delivered after the time constraint specified, the outcome is defined as timed-out.

The placing of audit data into the database can be performed either with or without human involvement. Possible outcomes are success or failure. Failure results when any portion of the data cannot be placed into the database due to an error in the database itself. An error may also occur due to physical storage failure. If data is input into the database correctly, but not within the time specified, a time-out occurs.

The querying of audit data also applies to the Database. Its output is the result returned from the query. Outcomes are successful, erroneous data, or timed-out. The querying service is successful if the results are correct/expected and done within the time constraint specified. An erroneous outcome occurs if the results returned do not match the criteria specified in the query. A time-out results if the results are returned correctly, but not within the time allotted. The use of time constraints are optional in all tests.

The data encoding method component is addressed in the following section.

3.5 Workload

3.5.1 ASCII. Workloads are categorized into three types: "ASCII", "XML", and "binary XML". The ASCII log is a copy of actual logs generated from an AFIT Linux operating system's "Linux Audit Subsystem", i.e., Linux's kernel logs, which is configured as a cluster, running Red Hat Enterprise Linux 4/5. The audit content is from mid-February to early-June 2010 and transformed into an ASCII form by using the Linux command-line tool "ausearch". This ASCII form is an interpreted version of the default content, interpreting user IDs, and other number data into human-

readable text/names. The command presented below uses the option "-i" to interpret, and returns content between the interval specified by the start-time and end-time, respectively.

```
ausearch -i -if <filename> -st DD/MM/YYYY hh:mm -te DD/MM/YYYY hh:mm > <outputfile>
```

A PERL script automates the process of taking raw logs and using the ausearch tool to also break the logs into individual files per time-unit (i.e., daily, halfday, hourly). This results in 102-daily files, 205-halfday files, and 2,446-hourly files.

These ASCII files are used in the computer, networking, and database components. However, the ASCII files as they are cannot be placed directly into the database. They must be transformed into a format that the database supports, which is comma-delimited columns and rows (.csv files). This transforming is outside the experiment; thus tools, time and other resources used are ignored. However, the process uses the XML database itself as the vehicle to transform the ASCII to CSV. The ASCII in the experiment is not directly transformed, and the XML content is transformed into the CSV files. This would not be done in the real-world, although it was done here, to aid in determining and laying out the ASCII-CSV / relational-database equivalence. The "conversion time" is not included in the processing time discussed in Section 3.6.

3.5.2 XML. The ASCII files are analyzed to see how the information is related and interconnected, using a document about Linux Audit Logs from [IBM08]. Based on this an XML Schema file is created such that information with more than one occurrence is placed in an element of its own versus an attribute which can only have one value. Once the the Schema is done, another PERL script creates XML equivalent logs conforming to the Schema.

These XML files are used in the computer, networking, database, and data coding components.

3.5.3 Binary XML/EXI. The binary XML is the binary encoded form of an XML log. The XML binary encoding/decoding (codec) is based on the "candidate" (since it has not been ratified) standard created by the same organization that created XML, The World Wide Web Consortium. The data encoding component tested is the tool that creates Binary XML or EXI files. This tool is a Java-coded program using different coding modes, options (i.e. preservation options), and either the inclusion or non-inclusion of an XML Schema.

This workload type is used in the computer and networking components.

3.6 Performance Metrics

The first system metric is a storage metric, the amount of hard drive space used on the computer for logging measures storage efficiency.

The network communications metric is throughput, which consists of bits per second. This determines the feasibility of an XML log scheme; therefore, it is an important metric.

The database component uses in-process time, as it stores audit data in the database. The in-process time determines whether placing ASCII log data into a relational database is faster than XML log data placed into an XML database. Retrieving the queried information in a timely manner is also important. Thus, the time to query is also recorded.

3.7 System Parameters

The amount of memory in the system affects the performance of the Audit Logging System.

The hard drive capacity of a computer largely determines the capacity of a system and, as a result, influences the services provided and their capabilities.

The type of processors can affect the performance significantly. If a processor is tailored to a particular often used operation, then it is highly likely that the "tailored" processor will outperform a general purpose processor.

The operating system is typically general purpose and is seldom "specialized" for desktop computers and servers. However, some operating systems are better suited to particular applications. For example, Unix/Linux systems lend themselves to networking as web servers, while Windows systems are oriented toward business applications, such as marketing.

Swap space/paging is another system parameter which, if not controlled, may cause unforeseen or erroneous results. Since without sufficient memory, code or data may be placed on the hard drive rather than in memory, swapping will cause a delay in execution.

The number of processes executing on a computer significantly affects performance and the ability to perform controlled tests, as processes may compete for the CPU and interrupt the test process.

The network has fewer system parameters to control. A major system parameter, however, is the "type of network", such as Token Ring, Ethernet, etc. The choice of a network protocol can affect throughput, in addition to correctness of the datum sent from one system to another. The type and number of network services utilizing the network affects performance since bandwidth is shared, resulting in a greater likelihood of corruption as collisions can occur.

The database parameters consist of the database management system software used, indexing of the database, and how the database is configured. The choice of the vendor of a database management system influences performance due to the vendor's particular specialties. This has been shown for relational databases tested against relational database benchmarks [Gra93]. Indexing is enabled, which affects audit service because additional work is done by a database system. This indexing is necessary in order for the experiments to execute reasonably in seconds versus hours;

Table 3.1: Factors Table
The column titles are the factors and levels are below.

Workload Volume	Schema: Transfer Binary XML	Schema: XML Database
daily	With XML-Schema (BYTE)	With XML-Schema
halfday	Without XML-Schema (BYTE)	Without XML-Schema
hourly	With XML-Schema (bit)	
	Without XML-Schema (bit)	

also, it's more realistic. The rows and columns of tables created for the relational database are in at least First Normal Form [Kro02], which does not allow duplicate rows and thus the need to create additional tables for pieces of data that have one-to-many or many-to-many relationships. This follows the basic layout and configuration of tables in a relational database that is to store log data. A relational-database schema is in appendix A.1.1, which contains the create table commands used for each table with their respective columns' data type.

3.8 Factors

Among the workloads, there are several factors that affect different resource usage or performance. It is expected that when using XML Schema to derive binary XML from an XML log, it will be created more compactly than when not using XML schema. The binary XML created with an XML schema is dependent upon that schema. Another factor that affects file size results is the coding mode (BYTE or bit). Bit mode is certainly to be more compact than BYTE because of its Huffman-like encoding and that it is not aligned to bytes as BYTE coding is. When "aligned" to a byte, if for example the data is encoded to 9 bit, then it would require 2 bytes. There are some trade-offs with each coding mode for other resources. When transmitting across the network, it should be transmitted in a shorter amount of time because of its compactness.

Regarding the XML database, there are no implementations at this time that import/export from a binary XML file. Therefore, the XML log itself is used, compared with versus without the schema.

Since the query language XQuery was designed for use with XML Schema, it is expected to perform better with the schema with a shorter response time, although loading the data into the database itself is expected to be slower.

Another factor that affects performance is the workload size. It is expected the volume of log data transmitted across a network and stored in a database could vary widely. Therefore, different workload sizes are tested, namely daily, halfday, and hourly. Each unit is a file made of an audit content for that day, halfday, or hour, respectively.

3.9 Evaluation Technique

The evaluation technique used is measurement. However, there is no attempt to replicate a "real-world" network. Rather, minimal background traffic is used. The measuring of the network transmission uses FTP since it is one of the simplest network protocols. "rcp" would be simpler, but systems do not have them by default, nor is it used because of security risks. Using an encrypted variant would have too much overhead, resulting in exaggerated transmission times.

The equipment used consists of three desktop computers and 100 megabit Ethernet network cables and a switch. Each computer has 2 *Intel Duo CPU T2450* at 2.0 gigahertz processors with 2 gigabytes of memory and 146.5 gigabytes of hard drive capacity using the "ext3" file system. The operating system is Linux Ubuntu 9.10 (kernel *2.6.31-14-generic*) configured with minimal services for the least impact on the system performance. The swap space for Linux is not used, in order to control memory usage.

The databases used for the ASCII log system and XML log system are "PostgreSQL 8.4.2" and "Berkeley DB XML 2.5.16", respectively. The databases are on dedicated computers to avoid interference with each other, which could result in biased measurements. They are configured with indexing in order for the experiments to execute reasonably in seconds versus hours. It is also more realistic.

The two database systems, in addition to being different types, do not have the same focus. The PostgreSQL is designed and setup as a stand-alone server in client/server architecture. This allows it to start up and prepare and optimize itself before receiving any connectivity and processing queries. The Berkeley is an embedded system. It is designed to be integrated into a individual application. Thus, it does not have the opportunity to prepare or optimize itself in advance. It performs the query when the application does the querying. This is noted, but the PostgreSQL is used because it supports some querying features that are equivalent to that of the XQuery language for an unbiased comparison.

The Java library used to program the tool to encode and decode an XML log into binary XML is "exificient-0.4". Although many options are used, only a specific combined-set is analyzed, *Preserve Namespace Prefixes* & *Preserve schemaLocation*, since it retains those properties that a system or human would use. Workloads are stored and "originate" on the third computer.

To validate the measured results, analytic analysis is used. The size of hard drive is relatively static and an analytic technique is used to estimate typical, current, and future or real-world usage.

3.10 Experimental Design

The experimental design is partial factorial; that is, a subset of combinations are tested: binary XML with schema and a medium volume versus binary XML without schema and a halfday volume versus XML log with schema and halfday volume versus XML log without schema and halfday volume. To gather reasonable data for statistical analysis, experiments are performed at least 3 times each.

3.11 Methodology Summary

This experiment compares the two ASCII log and XML log systems. It is expected that the ASCII log will use fewer resources on the computer and have overall

better performance for the network and database than the XML log. However, the XML Binary log should do better than ASCII logs in hard drive space and shorter network transmission time.

IV. Analysis of ASCII, XML, & EXI

4.1 Introduction

Analysis moves through each component in the order of computer, encoding, network, and database. Beginning with the computer, the simple metric of hard drive space is compared for ASCII, XML, and the different variants of EXI per workload type from largest to smallest (daily, halfday, hourly). The variants of EXI are in the order encoded using BYTE mode without Schema, BYTE mode with schema, bit mode without schema, and bit mode with schema. Since the number of files of each type per workload type is the same (e.g., 102 daily ASCII files, 102 daily XML files, 102 daily EXI BYTE w/o schema files, etc.) and all these test files are greater than 100, an average (mean) of sizes are taken. It can also be seen that all relevant results collected on the respective tests of components that involve files (i.e. computer, encoding, network) are a statistical sample and not a population because logs are infinite over time and thus always a sample. Therefore, all statistical analyses done are samples, with a confidence level of 95% where necessary.

In analyzing the codec (i.e. encoding & decoding) component times are compared among BYTE with schema, BYTE without schema, etc. Then the correlation of file size to time is analyzed. Between the encoding & decoding, encoding is examined first, then decoding.

The second to last component analyzed is the network. The similar comparison cycle of ASCII, XML, EXI, with its variations BYTE without schema, etc., is analyzed for transmission time.

The last component analyzed is the database. The analysis is divided into two parts, the first being loading logs into the database, and the other being the querying of data. ASCII are loaded logs into the relational database, and XML logs are loaded into the XML database, but having the Schema factor with and without a schema.

The loading of the CSV files into the database, is executed on the server backend, using its “copy” command. This command is not to be confused with the identically

Table 4.1: File Sizes Averages Summary (Kilobytes)

ASCII	XML	EXI		
			w/o	w/
Daily	2211.58	3463.56	BYTE	674.49	614.7
			bit	517.27	425
Halfday	1412.14	2215.72	BYTE	430.39	393.54
			bit	330.1	272.17
Hourly	118.35	185.23	BYTE	35.95	32.82
			bit	27.49	22.46

named command used on the client side or frontend. The copy command is executed per file and placed in its appropriate table, i.e., a 1-to-1 file to table correspondence. Therefore, in order to load a collection of CSV files into their tables that is equivalent to the corresponding XML document, the collection of CSV files are executed within a SQL transaction block. This makes it an all-or-nothing single process, made of a series of small "copy" processes.

The loading of the XMLDB is the execution of the "putDocument" command. The command is written as:

```
putDocument <documentnameInDatabase> <XMLfileToLoad> f
```

The *documentnameInDatabase* is the name of the XML file once in the database. The "f" informs the database that the second argument, <*XMLfileToLoad*>, is a file, and not the result of another query or a string containing XML.

The analyzing of the querying experiments involves only two unrelated queries of the four executed on each database, due to limited research time for analysis. The latter query is more complex and is the rationale for the RDBMS used. In both parts, time is analyzed. And because there are differences initial and subsequent trials/rounds (R1, R2, R3), the first and second are looked at more closely.

4.2 Computer

4.2.1 Comparing file sizes. A summary of file sizes' averages are presented in Table 4.1, which also includes the XML binary forms with the two coding modes

without and with the schema used. The Daily is an average of 102 files, Halfday of 205 files, and Hourly of 2446 files. Kilobytes and Megabytes are shown when appropriate for each instead of the raw bytes for more compact numbers, although calculations are in bytes.

XML is shown to be larger than ASCII across all workloads, as is expected. The EXI files in all combinations presented are smaller than XML, as the binary encoding is designed to do, and are smaller than the ASCII. At a glance, the use of an XML Schema does result in smaller file sizes, and bit mode does use less than the byte mode since the data does not need to be aligned. Thus the most compact files are bit mode encoded with a schema. It is also shown in each column that the size of the workload (daily versus hourly) is a contributing factor to file size across all types.

The file size ratios with ASCII as base Table 4.2 shows that XML should not be used for any term of storage other than short since it will consume hard drive space over 1.6 times the rate that ASCII. Coding should be used for XML if is is to be stored in files for any longer length of time. For the EXI, as the workload size is smaller, there is less to be converted to binary, resulting less saving in hard drive resource, although the reduction remains significant.

T-tests are used to verify that there is a statistically significant difference in file sizes between the non-use and use of a schema across all workloads. If the confidence interval includes zero, they are not statistically different; otherwise they are at the p-value level of significance. As is seen in Table 4.3, the p-values are quite small for both BYTE and bit among all workloads. Thus, the null hypothesis, the mean difference between with a Schema and without Schema is zero, is rejected. The confidence intervals not including zero support this same conclusion. A similar conclusion for BYTE and bit mode encodings is shown in Table 4.4.

Table 4.2: File size (avg) Ratios: ASCII as base

	XML	EXI		
			w/o	w/
Daily	1.621	BYTE	0.345	0.302
		bit	0.271	0.202
Halfday	1.636	BYTE	0.353	0.304
		bit	0.278	0.203
Hourly	1.629	BYTE	0.434	0.321
		bit	0.357	0.22

Table 4.3: File sizes EXI Schema significant difference T-test

	datatype	BYTE	bit
Daily	t-value	-3.3219	-3.845
	degrees of freedom	101	101
	confidence level	95%	95%
	p-value	0.0012	0.00021
	confidence interval	(-97774.31, -24660.20)	(-143231.88, -45737.06)
Halfday	t-value	-4.6942	-5.1697
	degrees of freedom	204	204
	confidence level	95%	95%
	p-value	4.9×10^{-06}	5.6×10^{-07}
	confidence interval	(-53575.77, -21881.98)	(-81944.31, -36696.16)
Hourly	t-value	-11.4589	-10.2089
	degrees of freedom	2445	2445
	confidence level	95%	95%
	p-value	$< 2.2 \times 10^{-16}$	$< 2.2 \times 10^{-16}$
	confidence interval	(-3750.69, -2654.57)	(-6137.13, -4159.37)

Table 4.4: File sizes EXI Coding-mode significant difference T-test

	datatype	without Schema	with Schema
Daily	t-value	3.7594	3.9469
	degrees of freedom	101	101
	confidence level	95%	95%
	p-value	0.00029	0.00015
	confidence interval	(76037.15, 245931.56)	(96618.76, 291884.38)
Halfday	t-value	4.5157	4.7362
	degrees of freedom	204	204
	confidence level	95%	95%
	p-value	1.1×10^{-05}	4.1×10^{-06}
	confidence interval	(57857.73, 147537.92)	(72548.47, 176029.90)
Hourly	t-value	8.9231	8.845
	degrees of freedom	2445	2445
	confidence level	95%	95%
	p-value	$< 2.2 \times 10^{-16}$	$< 2.2 \times 10^{-16}$
	confidence interval	(6757.62, 10564.25)	(8255.08, 12958.03)

4.3 Codec Methods

4.3.1 Codec Time & Memory. The execution time is the difference of end-time − start-time, retrieved after the end and prior to the start of the execution. The times retrieved were system-calls that returned the number of milliseconds from January 1, 1970 UTC.

The collecting of memory usage for the encoding and decoding processes was orchestrated by clearing all garbage memory to form a clean baseline of memory usage prior to executing the encoding / decoding action. Since Java has no function that returns the amount of memory that the process itself is using, it was determined by taking the difference of the total memory of the runtime in the Java Virtual Machine (JVM) and subtracting the amount of Free memory of the runtime in the JVM. Since there are no other processes executing within the JVM other than the codec, this provides an accurate memory recording at the heart of the encoding and decoding functions. However, due to limited research time, memory usages are shown for encoding in Table 4.6 and decoding in Table 4.7, but not analyzed.

Table 4.5: Encoding times Average Summary (Seconds)

		Encoding Time	
		w/o	w/
Daily	BYTE	2.341	2.211
	bit	1.910	1.695
Halfday	BYTE	1.506	1.430
	bit	1.228	1.091
Hourly	BYTE	0.135	0.140
	bit	0.108	0.110

Table 4.6: Encoding memory Average Summary (Kilobytes)

		Encoding Time	
		w/o	w/
Daily	BYTE	14966.63	16062.90
	bit	15051.02	14806.50
Halfday	BYTE	10674.51	12871.77
	bit	10683.82	12238.27
Hourly	BYTE	3423.999	4611.014
	bit	3423.984	4610.996

Table 4.7: Decoding memory Average Summary (Kilobytes)

		Encoding Time	
		w/o	w/
Daily	BYTE	21739.32	24414.55
	bit	21693.05	24482.91
Halfday	BYTE	14377.95	17622.66
	bit	14356.90	18394.41
Hourly	BYTE	4489.800	5362.809
	bit	4488.068	5353.718

4.3.2 Time. As shown in Table 4.5, the use of the schema for the two larger files does result in a shorter encoding time. However, when encoding small files hourly, the schema becomes a liability if time is the variable to reduce. This is likely due to more time spent analyzing content to optimize compactness, but little or nothing is gained, resulting in wasted time. Among the larger files, there is a statistical difference in time between the use and non-use of a schema for both modes of coding, as shown in Table 4.8. However, for the hourly workload, because the times on average are close, it is not necessarily statistically different either. It is shown to be for byte mode encoding, but not for bit mode. The p-value is in the region of the confidence interval, so it cannot be rejected and is to be accepted that bit mode encoding with or without a schema are the same in time. It is also observed that the confidence interval includes zero, so the same conclusion is reached.

In comparing time usages at the coding mode level, again in Table 4.5, the bit mode is shorter than byte mode encoding for all the workloads for both without and with schema. It was presumed that using the more compact bit mode would require more time for the process to encode source data that is generally byte-oriented to bits, versus source byte-oriented data to byte data that is less change and less work, resulting in less time. The T-tests verifies that there are statistical differences in time, shown in Table 4.9, among workloads and schema usage or not.

It is shown in Table 4.5 that the time decreases as the workloads used are smaller. Correlation tests using Pearson are used to verify that, shown in Table 4.10. Having all results being very close to +1, it is concluded that there is a strong positive correlation of file sizes affecting the encoding time.

The decoding times shown in Table 4.11 are much smaller than the encoding times. This is important in that the receiving end doing the decoding is able to equal the encoding time or use less time, so that it is not overwhelmed. When decoding, more time is needed when using a schema versus without a schema, which is the reverse of the encoding process. The bit mode decoding is often uses less time than

Table 4.8: Encoding time Schema significant difference T-test

	datatype	BYTE	bit
Daily	t-value	2.904	3.4217
	degrees of freedom	101	101
	confidence level	95%	95%
	p-value	0.0045	0.0009
	confidence interval	(0.041 , 0.218)	(0.09, 0.34)
Halfday	t-value	3.6118	4.7628
	degrees of freedom	204	204
	confidence level	95%	95%
	p-value	0.00038	3.6×10^{-06}
	confidence interval	(0.034, 0.116)	(0.081, 0.195)
Hourly	t-value	-6.1166	-1.198
	degrees of freedom	2445	2445
	confidence level	95%	95%
	p-value	1.1×10^{-09}	0.2310
	confidence interval	(-0.0078, -0.004)	(-0.0046, 0.0011)

Table 4.9: Encoding time Coding-mode significant difference T-test

	datatype	without Schema	with Schema
Daily	t-value	3.891	3.9931
	degrees of freedom	101	101
	confidence level	95%	95%
	p-value	0.00018	0.00012
	confidence interval	(0.21, 0.65)	(0.26, 0.77)
Halfday	t-value	4.4152	4.7984
	degrees of freedom	204	204
	confidence level	95%	95%
	p-value	1.6×10^{-05}	3.09×10^{-06}
	confidence interval	(0.15, 0.4)	(0.2, 0.48)
Hourly	t-value	8.6315	8.8115
	degrees of freedom	2445	2445
	confidence level	95%	95%
	p-value	$< 2.2 \times 10^{-16}$	$< 2.2 \times 10^{-16}$
	confidence interval	(0.02, 0.03)	(0.02, 0.04)

Table 4.10: Encoding time to file size correlation (Pearson)

Encoding Time = file size correlation			
		w/o	w/
Daily	BYTE	0.9997	0.9999
	bit	0.9995	0.9996
Halfday	BYTE	0.9996	0.9996
	bit	0.9997	0.9993
Hourly	BYTE	0.9965	0.9977
	bit	0.9994	0.9989

Table 4.11: Decoding times Average Summary (Seconds)

Decoding Time			
		w/o	w/
Daily	BYTE	0.237	0.269
	bit	0.249	0.298
Halfday	BYTE	0.170	0.187
	bit	0.159	0.173
Hourly	BYTE	0.02126	0.03738
	bit	0.02066	0.03736

the byte decoding except for the largest workload, daily, regardless of schema. This may be because more work is required to reconstruct the larger content. T-tests are used to verify the difference in schema results, shown in Table 4.12. As can be seen regarding the schema in the T-tests, most of the time when taking the schema into consideration is statistically different, except for the bit mode decoding on the large daily logs, where zero is included in the confidence interval. However, when looking at the statistical difference of coding modes using T-tests shown in Table 4.13, many times there is not a difference, except for the halfday workload. However, the p-value, relative to other previous statistical analyses, is not that small. It should be cautiously stated there are statistical differences in time between using BYTE and bit mode decodings regardless of schema usage. There is a strong correlation of file size to decoding time with all results shown in Table 4.14 close to +1.

Table 4.12: Decoding time Schema significant difference T-test

	datatype	BYTE	bit
Daily	t-value	-4.1575	-1.6928
	degrees of freedom	101	101
	confidence level	95%	95%
	p-value	6.758×10^{-05}	0.09358
	confidence interval	(-0.047, -0.017)	(-0.1064, 0.0084)
Halfday	t-value	-3.45	-3.9414
	degrees of freedom	204	204
	confidence level	95%	95%
	p-value	0.00068	0.00011
	confidence interval	(-0.0276, -0.0075)	(-0.0216, -0.0072)
Hourly	t-value	-14.664	-36.5649
	degrees of freedom	2445	2445
	confidence level	95%	95%
	p-value	$< 2.2 \times 10^{-16}$	$< 2.2 \times 10^{-16}$
	confidence interval	(-0.018, -0.014)	(-0.018, -0.016)

Table 4.13: Decoding time Coding-mode significant difference T-test

	datatype	without Schema	with Schema
Daily	t-value	-1.3751	-0.9912
	degrees of freedom	101	101
	confidence level	95%	95%
	p-value	0.1721	0.324
	confidence interval	(-0.029 , 0.0053)	(-0.087 , 0.029)
Halfday	t-value	2.3479	2.5669
	degrees of freedom	204	204
	confidence level	95%	95%
	p-value	0.0198	0.011
	confidence interval	(0.0018, 0.0207)	(0.0033, 0.0254)
Hourly	t-value	0.5484	0.0463
	degrees of freedom	2445	2445
	confidence level	95%	95%
	p-value	0.583	0.96
	confidence interval	(-0.0015, 0.0027)	(-0.0008 0.0008)

Table 4.14: Decoding time to file size correlation (Pearson)

Encoding Time = file size correlation			
		w/o	w/
Daily	BYTE	0.997	0.996
	bit	0.996	0.924
Halfday	BYTE	0.995	0.996
	bit	0.993	0.996
Hourly	BYTE	0.827	0.979
	bit	0.976	0.982

4.4 Network Transfer

Due to a misconfiguration of FTP commands for the Hourly bit mode binary XML, some results are missing from the hourly set of the network component. Any analysis of that set of results would be incorrect. Therefore, bit coded hourly related results and analysis is excluded. Correcting and regenerating the correct results cannot be done because of limited time.

Table 4.15 shows, similar to the file sizes presented earlier, that with schema is a shorter time than without schema for all workloads shown. It is also shown that bit mode can be a shorter time, but does not appear to be so on average for daily and halfday workloads. For the daily, the bit mode with the schema becomes a liability. When using ASCII as the base for ratios and presented as averages in Table 4.16, it is clear the XML when transmitting does worse than ASCII, as is expected. However, when compared to EXI (ASCII as base), EXI performs better than ASCII, although it may only be a little bit shorter in time, as is the case for workload halfday used with bit encoding and with a schema.

Since there appears to be no consistent pattern, a t-test determines if there are any differences in using a schema, shown in Table 4.17, and if there is any difference between modes, shown in Table 4.18. As can be seen in workloads daily and halfday sets of t-tests, the daily workload use of a schema and bit mode, respectively, is each statistically different and thus better than without a schema on Byte mode. On the other hand, the halfday t-tests all show the confidence interval including zero,

Table 4.15: Transmission times Averages Summary (seconds)

	ASCII	XML	EXI		
				w/o	w/
Daily	0.2008	0.3215	BYTE	0.0556	0.0375
			bit	0.0544	0.0479
Halfday	0.1329	0.2102	BYTE	0.0556	0.0375
			bit	0.0394	0.0351
Hourly	0.0121	0.0257	BYTE	0.0042	0.0038
			bit	UNKNOWN	UNKNOWN

Table 4.16: Network Times (avg) Ratios: ASCII as base

	XML	EXI		
			w/o	w/
Daily	1.5591	BYTE	0.6401	0.5867
		bit	0.6961	0.7230
Halfday	1.5472	BYTE	0.7211	0.3953
		bit	0.7341	0.9924
Hourly	2.1219	BYTE	0.3491	0.3128
		bit	UNKNOWN	UNKNOWN

concluding that neither the use of a schema or bit packed is better. The hourly, for byte mode, shows the schema factor is statistically different.

To confirm that EXI does transmit in a shorter time than ASCII, a t-test compares to Byte mode without Schema, since it is the least efficient of EXI transfers. As shown in Table 4.19, the confidence intervals for all workloads do not include zero. Thus there is a statistical difference, meaning that EXI is verified to transmit data in shorter time.

There is a scatter plot with the regression line drawn to show the strong correlation of file size to transmission time, as are shown in Figures 4.1 and 4.2.

Table 4.17: Transmission times Schema significant difference T-test

	datatype	BYTE	bit
Daily	t-value	3.0684	2.6429
	degrees of freedom	101	101
	confidence level	95%	95%
	p-value	0.002763	0.00953
	confidence interval	(0.002, 0.009)	(0.0016 , 0.0114)
Halfday	t-value	2.1292	0.5211
	degrees of freedom	204	204
	confidence level	95%	95%
	p-value	0.03444	0.6029
	confidence interval	(0.001, 0.03)	(-0.01 , 0.02)
Hourly	t-value	4.143	UKNOWN
	degrees of freedom	2445	2445
	confidence level	95%	95%
	p-value	3.544×10^{-05}	UNKNOWN
	confidence interval	(0.00023, 0.00065)	UNKNOWN

Table 4.18: Transmission time Coding-mode significant difference T-test

	datatype	without Schema	with Schema
Daily	t-value	3.0261	2.5965
	degrees of freedom	101	101
	confidence level	95%	95%
	p-value	0.003143	0.01082
	confidence interval	(0.0042, 0.0204)	(0.0032, 0.0236)
Halfday	t-value	1.8379	0.2748
	degrees of freedom	204	204
	confidence level	95%	95%
	p-value	0.06753	0.7837
	confidence interval	(-0.001, 0.033)	(-0.01, 0.02)

Table 4.19: Transmission time ASCII versus EXI significant difference T-test

	datatype	value
Daily	t-value	3.8722
	degrees of freedom	101
	confidence level	95%
	p-value	0.0001916
	confidence interval	(0.065, 0.20)
Halfday	t-value	3.4844
	degrees of freedom	204
	confidence level	95%
	p-value	0.0006037
	confidence interval	(0.03, 0.12)
Halfday	t-value	9.3186
	degrees of freedom	2445
	confidence level	95%
	p-value	$< 2.2 \times 10^{-16}$
	confidence interval	(0.0062, 0.0095)

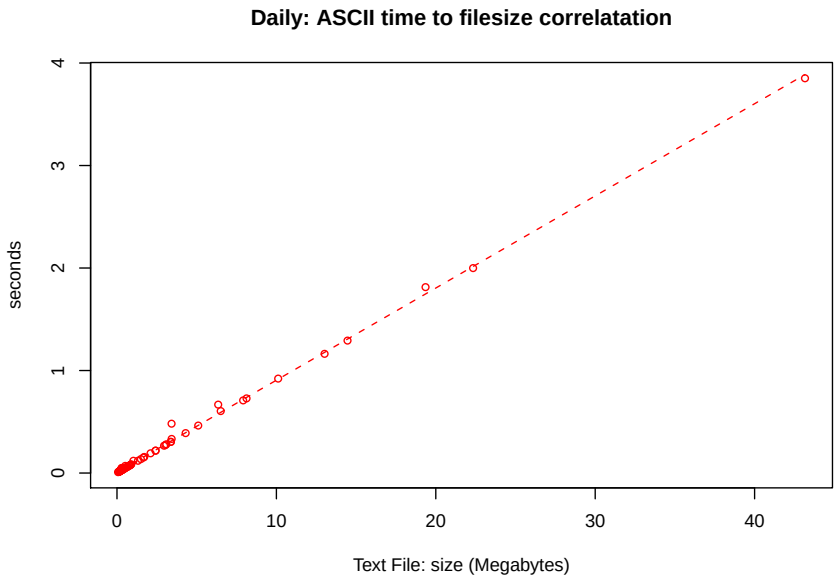

Figure 4.1: Daily: ASCII scatter plot time to file size correlation

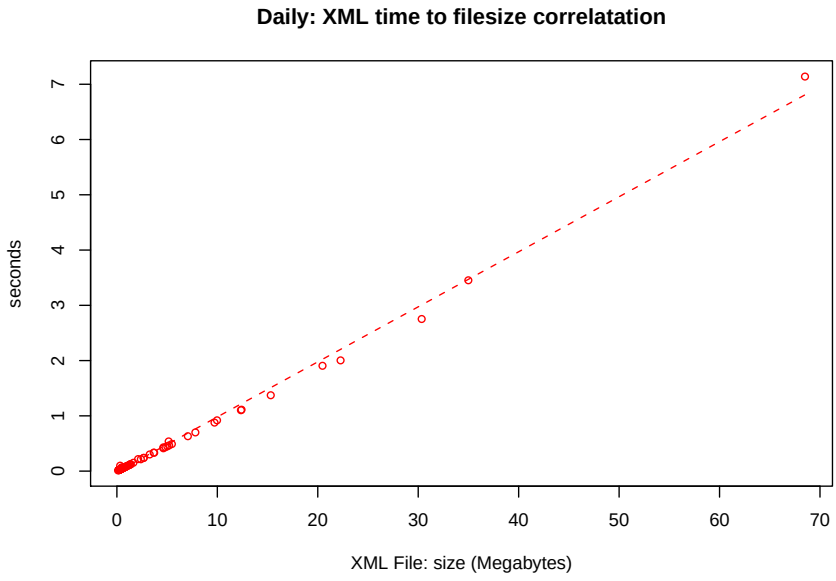

Figure 4.2: Daily: XML scatter plot time to file size correlation

Table 4.20: Loading avg Time (sec)

	RDBMS	XMLDB w/o Schema	XMLDB w/ Schema
Daily	2.583	11.398	12.026
Halfday	1.890	7.696	7.651
Hourly	0.148	0.605	0.610

4.5 Database

4.5.1 Loading database. The loading time of each of the workloads that is made using the average time of each is seen in a summary in Table 4.20. Across workloads the RDBMS executes in less time than the XMLDB. RDBMS is a stand-alone client/server architecture which allows it to do preliminary setup and optimization before any connections or data are dealt with. The XMLDB is used in an embedded system and is only active as an application uses its internal database. Thus preliminary setup and optimization cannot occur.

When comparing the loading of XML with a schema versus without the schema, more often than not, using an XML Schema does result in a longer time to load the data with the expected delay in checking the data against schema compliance. The occurrence in which it did not take longer was loading the halfday set of data.

However, regarding the statistical significance of not using a schema in providing a shorter loading performance, this is not so for halfday and hourly workloads, as shown in Table 4.21. Their confidence intervals include zero. The daily workload marginally states it does, with the p-value being 0.01.

4.5.2 Querying database. The databases querying uses a series of query scripts with the appropriate language for each database system. Structured Query Language:2003 (SQL) is for the relational database, and XML Query Language (XQuery) for the XML database. Scripts are executed without human error and/or human inefficiency.

Table 4.21: Loading XMLDB Daily Schema significant difference T-test

	datatype	values
Daily	t-value	2.4354
	degrees of freedom	101
	confidence level	95%%
	p-value	0.01663
	confidence interval	(0.12, 1.14)
Halfday	t-value	-0.8308
	degrees of freedom	204
	confidence level	95%
	p-value	0.4071
	confidence interval	(-0.2 0.1)
Hourly	t-value	0.9492
	degrees of freedom	101
	confidence level	95%
	p-value	0.3426
	confidence interval	(-0.0056, 0.016)

4.5.2.1 Query: System's Arguments. One set of items queried, the "Arguments list" of the syscall, is recorded in the Linux logs. Another set of items queried is the filesystem path information in the audit log system, particularly only the third path item.

The results from the querying of the "arguments" shows that the XMLDB executed the query quicker, using less memory. Tables 4.22, 4.23, and 4.24 show query times, and Tables 4.25, 4.26, and 4.27 show memory usage. The shorter execution times occur after the initial run, which gives the XMLDB time to do some caching. If caching-size for the XML db is set to 0, the system would slow considerably, and would be less realistic. There is only one case where XMLDB uses more memory than the RDBMS, i.e., on the third run for hourly set of data. Graphical comparisons are presented in Figures 4.3 to 4.8, showing the first and second round of each dataset.

Below is the SQL code:

```
SELECT entry.*, syscall.*, args.*
FROM syscall, args, entry
WHERE syscall.entry_key = args.entry_key and args.entry_key = entry.entry_key;
```

Table 4.22: Querying System's Arguments avg Time (sec) Daily

	RDBMS	XMLDB w/o Schema	XMLDB w/ Schema
Round 1	29.653	30.044	10.019
Round 2	32.7115	6.033	5.997
Round 3	36.550	5.926	5.980

Table 4.23: Querying System's Arguments avg Time (sec) Halfday

	RDBMS	XMLDB w/o Schema	XMLDB w/ Schema
Round 1	51.444	41.601	41.685
Round 2	51.814	7.713	7.8995
Round 3	50.696	7.777	7.8149

As seen

The XQuery code follows

! The setting of namesapce in this context is XQuery syntax per se, but is Berkeley DB XML's command line tool's command.

```
setNamespace la "urn:xmlns:xmlschema:AFIT/logs/Linux/Audit"

let $args_in_sys := collection()/la:LinuxAuditLog/la:entry[./la:syscall/la:arguments]

return data($args_in_sys)
```

Table 4.24: Querying System's Arguments avg Time (sec) Hourly

	RDBMS	XMLDB w/o Schema	XMLDB w/ Schema
Round 1	56.776	75.198	74.782
Round 2	53.336	7.912	7.923
Round 3	55.564	7.809	8.003

Table 4.25: Querying System's Arguments Memory (MB) Daily

	RDBMS	XMLDB w/o Schema	XMLDB w/ Schema
Round 1	144.141	110.199	102.117
Round 2	144.141	110.457	107.102
Round 3	144.141	105.930	111.106

Table 4.26: Querying System's Arguments Memory (MB) Halfday

	RDBMS	XMLDB w/o Schema	XMLDB w/ Schema
Round 1	145.137	137.805	136.637
Round 2	145.016	133.013	133.801
Round 3	145.016	133.801	134.430

Table 4.27: Querying System's Arguments Memory (MB) Hourly

	RDBMS	XMLDB w/o Schema	XMLDB w/ Schema
Round 1	190.141	148.496	149.922
Round 2	184.086	148.512	152.516
Round 3	143.133	148.109	143.965

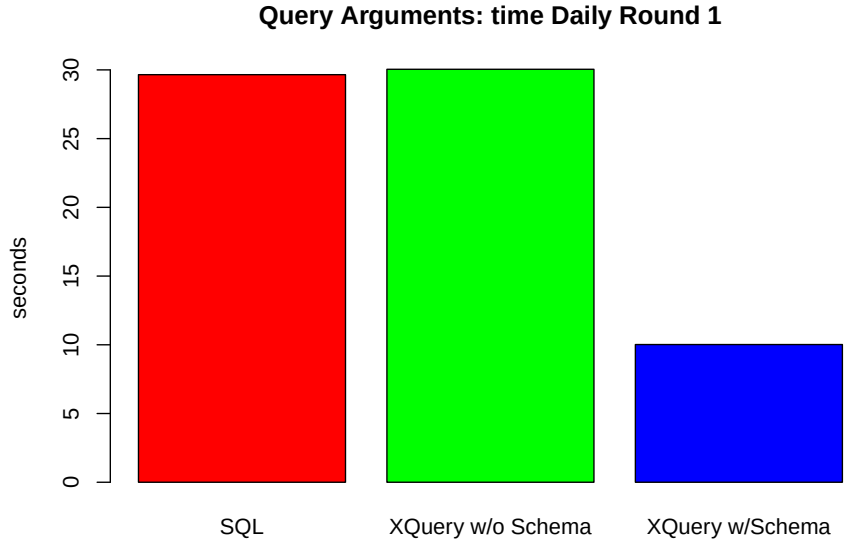

Figure 4.3: Querying for System Arguments: time Daily Round 1

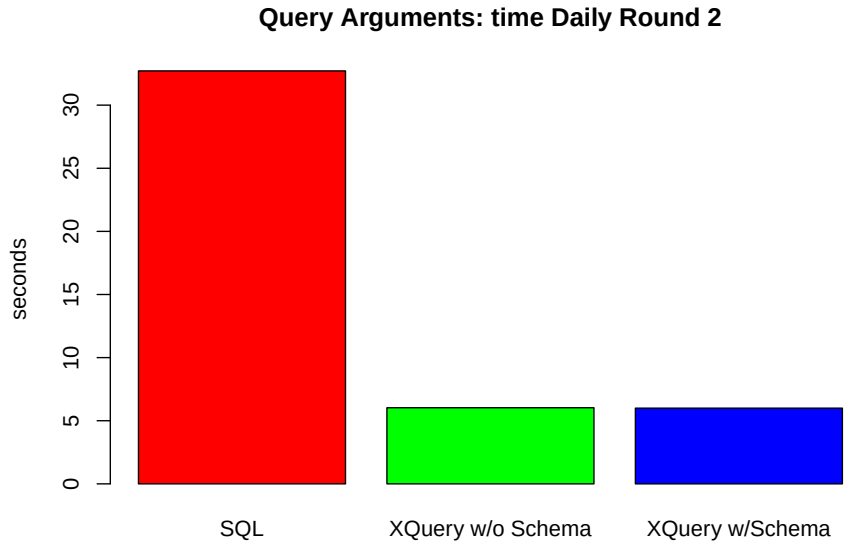

Figure 4.4: Querying for System Arguments: time Daily Round 2

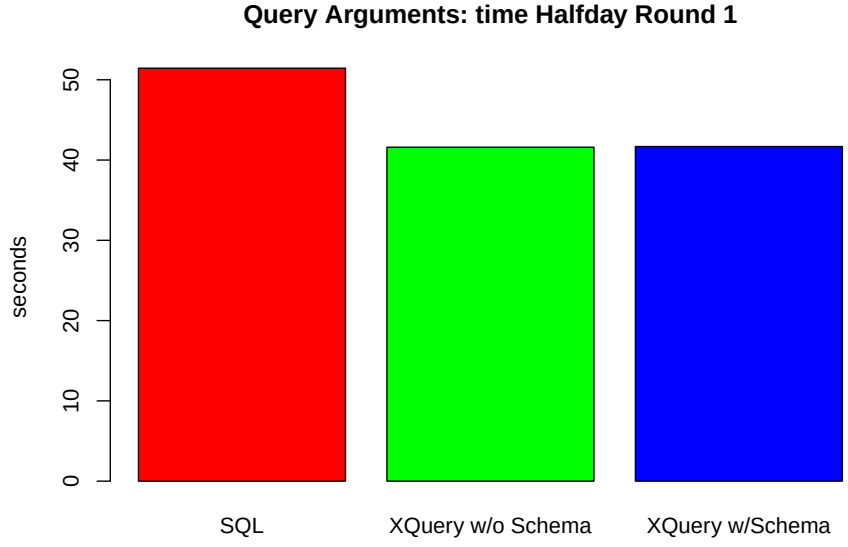

Figure 4.5: Querying for System Arguments: time Halfday Round 1

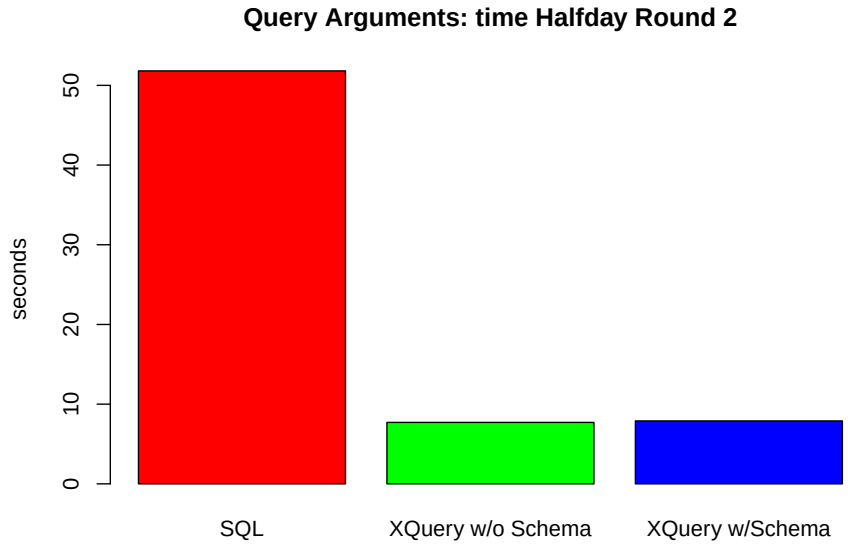

Figure 4.6: Querying for System Arguments: time Halfday Round 2

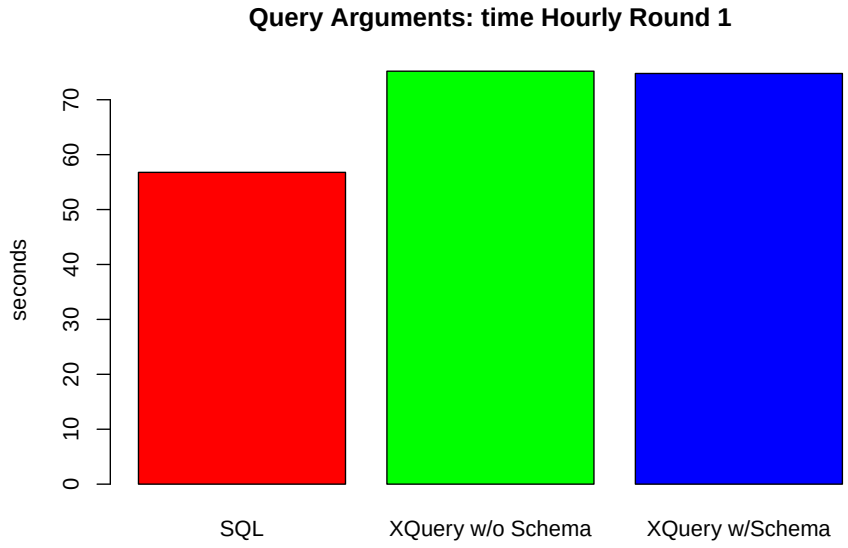

Figure 4.7: Querying for System Arguments: time Hourly Round 1

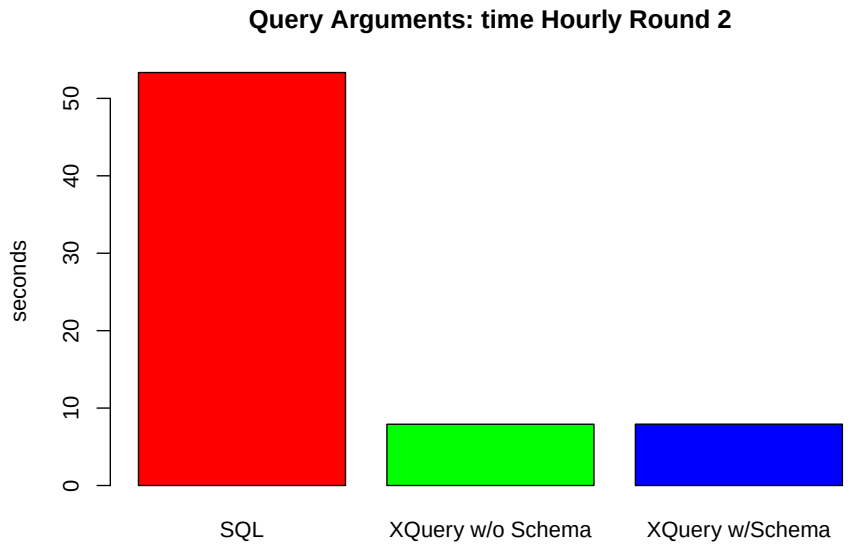

Figure 4.8: Querying for System Arguments: time Hourly Round 2

4.5.2.2 Query: System's third path. The significance of this query is to show that XQuery has "position" determination as part of the standard. The equivalent SQL statement, if the RDBMS does not support "Window functions", would require the use of a cursor to determine position. However, the use of cursors would put the RDBMS at a disadvantage, causing the results to be biased. The results from the querying of the "third path" are the complete opposite of the "arguments" query, shown in Table 4.28, 4.29, and 4.5.2.2. There is not a plateau for XMLDB as there was with the previous query. The memory of the RDBMS is lower than XMLDB for all runs of all datasets, shown in Tables 4.30, 4.31, and 4.32, yet there are some significant variations in memory for RDBMS, as well. For the initial round of the halfday dataset, it executes in 28 seconds, then drops to 3 seconds for the later rounds. A longer time for an initial round followed by shorter times does happen for the daily and hourly sets of data for this query, as well. Graphical comparisons are in Figures 4.9 to 4.14

Comparing the use of a schema or not, there are no consistent results. Sometimes the schema runs in a shorter time than without it. Other times it takes longer. Thus, there is no effect.

The query for the syscall information that has 3 or more "path" items associated to the syscall returns that syscall information but with only the third path and related entry information. The SQL code follows:

```
select *
from (SELECT row_number() over (partition by entry_key) positionNumber, path.*
FROM path) thirdPath natural join syscall natural join entry
where positionNumber = 3
```

The XQuery then follows, with the advantage of annotating the data presented is abridged:

```
let $entries_that_have_syscall_w_third_plus_paths :=
```

Table 4.28: Querying System's Third path avg Time (sec) Daily

	RDBMS	XMLDB w/o Schema	XMLDB w/ Schema
Round 1	3.754	68.351	68.747
Round 2	2.809	68.565	68.772
Round 3	2.843	68.334	68.924

Table 4.29: Querying System's Third path avg Time (sec) Halfday

	RDBMS	XMLDB w/o Schema	XMLDB w/ Schema
Round 1	28.378	83.726	84.243
Round 2	3.325	82.597	84.667
Round 3	3.354	84.350	86.780

```
  collection()/la:LinuxAuditLog/la:entry[./la:syscall/la:items/count(./la:path) >= 3]
let $thisResults := <result>{
for $thisentry in $entries_that_have_syscall_w_third_plus_paths
let $this_path := $thisentry//la:path[3]

return <entry>{$thisentry/@*} {
<syscall>{$thisentry/la:syscall/@*} {
$this_path
}<!--This is a stripped down version of the syscall information--></syscall>
}</entry>

}</result>

return $thisResults/*
```

captionQuerying System's Third path avg Time (sec) Hourly

	RDBMS	XMLDB w/o Schema	XMLDB w/ Schema
Round 1	3.298	86.371	84.012
Round 2	3.3049	85.119	83.290
Round 3	3.291	85.368	82.682

Table 4.30: Querying System's Third path Memory (MB) Daily

	RDBMS	XMLDB w/o Schema	XMLDB w/ Schema
Round 1	138.887	176.699	173.734
Round 2	95.711	167.301	173.992
Round 3	127.461	162.262	176.184

Table 4.31: Querying System's Third path Memory (MB) Halfday

	RDBMS	XMLDB w/o Schema	XMLDB w/ Schema
Round 1	181.996	179.922	181.223
Round 2	85.520	167.430	172.584
Round 3	95.820	181.223	181.223

Table 4.32: Querying System's Third path Memory (MB) Hourly

	RDBMS	XMLDB w/o Schema	XMLDB w/ Schema
Round 1	139.125	165.883	169.492
Round 2	139.125	181.223	181.223
Round 3	95.820	166.527	181.223

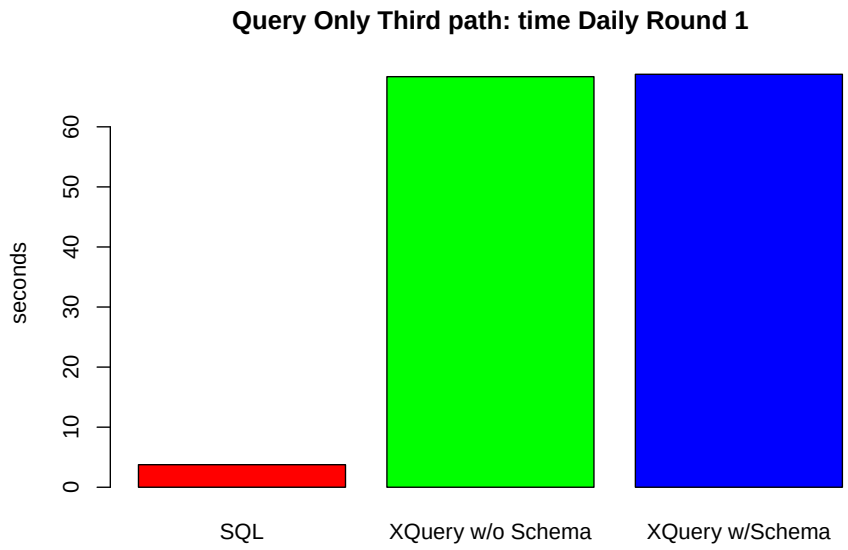

Figure 4.9: Querying for Only System's third path: time Daily Round 1

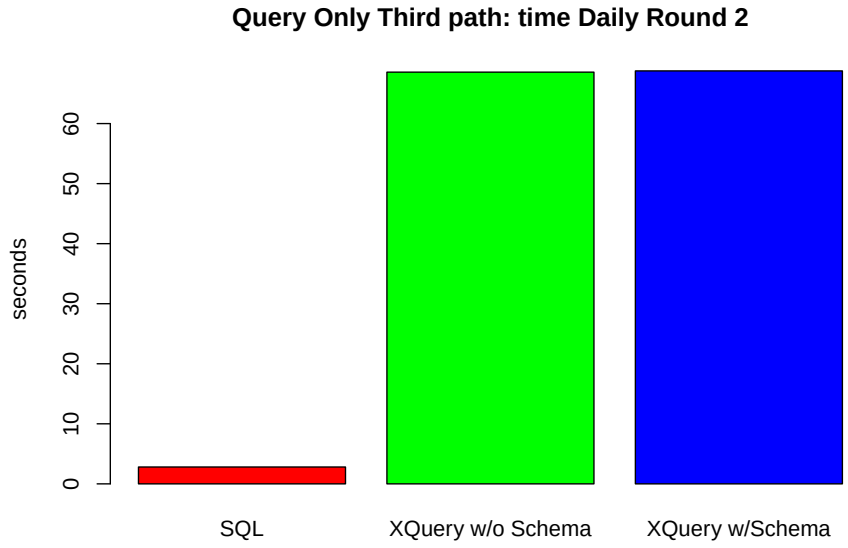

Figure 4.10: Querying for Only System's third path: time Daily Round 2

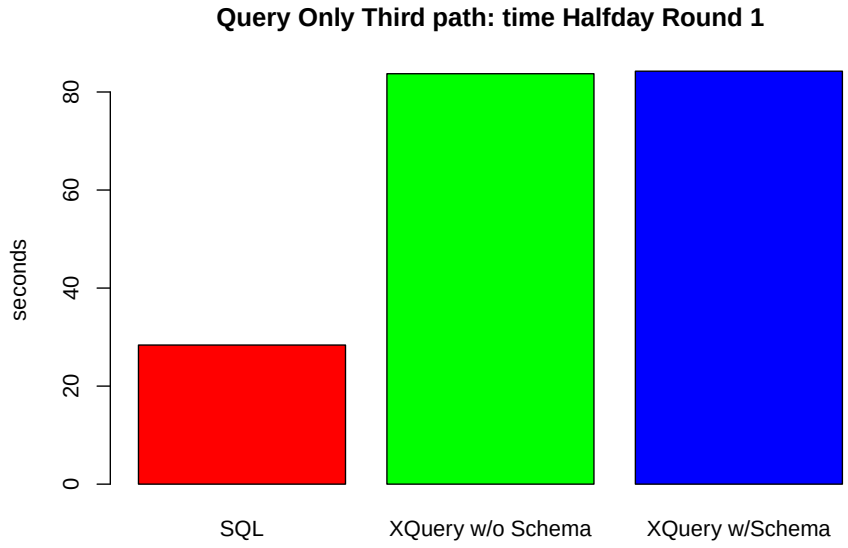

Figure 4.11: Querying for Only System's third path: time Halfday Round 1

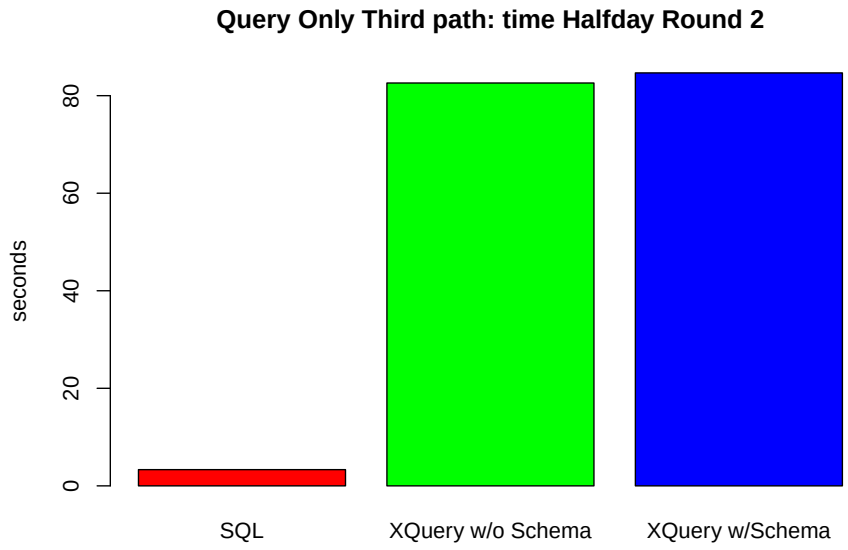

Figure 4.12: Querying for Only System's third path: time Halfday Round 2

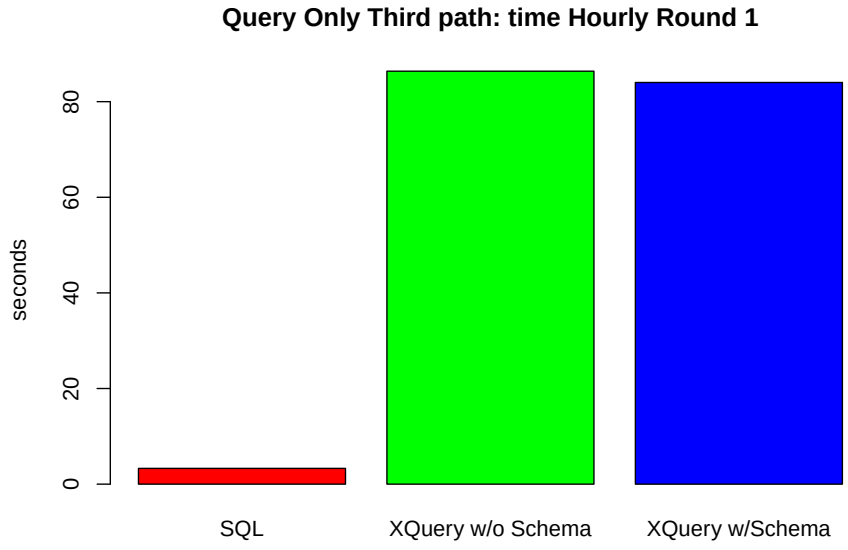

Figure 4.13: Querying for Only System's third path: time Hourly Round 1

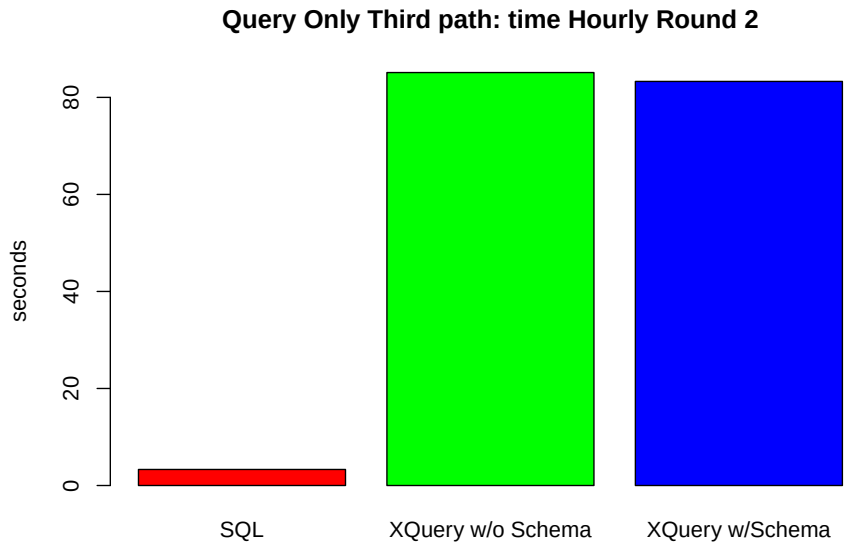

Figure 4.14: Querying for Only System's third path: time Hourly Round 2

V. Conclusions

5.1 XML Files

The use of XML files for the storage or transmission of data, and particularly for Linux Audit Log representation, is less efficient and should not be done unless there is a compelling reason to do so apart from log performance. It makes sense to use XML as storage if logs originate as XML and are to be used shortly, such as when placing data into an XML database. Using XML as the transport of data may be worthwhile if technologies already use XML, as may be the case for some web-services.

The testing of Efficient XML Interchange (EXI) or binary XML shows it does reduce the size of XML significantly and thus makes storing XML with XML's benefits of self-documenting data, extensibility, and almost universal application-supported technology without XML's excessive size feasible. It was anticipated that XML would be at least twice the size of the ASCII text file and that EXI would reduce the XML file to at most half the XML size, resulting in the binary XML being equal to or slightly larger than the original ASCII file. Instead, XML binary form resulted in being smaller than the ASCII, even when preserving the XML Namespace prefixes, and the SchemaLocation attribute information.

The only drawback to the use of EXI is a large memory footprint. Memory usage is reflective of the configurations applied, i.e., whether the encoding is packed at the bit level or aligned at the byte level, in addition to what is to be "preserved". Memory is also affected the reading in of XML Schema if used for the encoding / decoding processes. These are all factors in EXI's memory usage. Even so, memory usage may not be a concern as it is relatively inexpensive. The significant issue for XML is the network transmission and database query times.

5.2 Networking

There is a strong correlation of file size to time to transmit data, as expected. Since the XML files were the largest, followed by the ASCII, and lastly the EXI in

the different configurations tested, the transmission times were in proportion to their sizes.

5.3 Database

5.3.1 Loading data. It was expected that the XML database would be more efficient than the relational database as there is not anything that needs to be transformed/changed because the data of an XML database is the XML document itself. The relational database, on the other hand, type-cast the numeric text into integers among other data-types before placing it into the database. However, the relational database loaded the data in a shorter amount of time and used less memory. It is suspected that the relational database does so because it is designed as a standalone dedicated server, whereas the the XML database is not. Furthermore, the loading of the relational database was done from the server-side and not from the client side. In actual use, data would likely be loaded from the client side, which would have a longer processing time due to client overhead, as well as process connectivity and throughput limitations from the client to the server. The rationale for using the server-side was that it provides a more accurate comparison since the embedded XML database does not have a client/server overhead.

The time and memory taken to transform the ASCII text into a comma delimitated (.csv) file was not taken into consideration. There is no set method to convert ASCII text into .csv.

If XML databases support the new EXI "natively", then the loading time could be on par or better than that of RDBMS. "Natively" means no decoding of EXI to XML and that the XML database itself stores the XML in binary form.

5.3.2 Querying data. For the querying of the databases, neither is better than the other with respect to time or memory usage. It depends on how the query is written and how the database system processes, interprets, and optimizes the query in addition to what and how data is indexed. Even among the same class of databases

(i.e., relational) there are numerous disputes and comparisons of the performance of queries, as can be seen in the Transaction Processing Performance Council (TPC) database benchmarking.

5.4 Further Research

A more detailed comparison of the different configuration options and their usage of time and resources should be studied. The EXI code used in the experiment is not fully implemented and thus not everything in EXI such as dictionary-tables was tested. The implementation of a full ASCII log system versus XML logs system from start to finish would provide a more complete picture. The design, implementation and testing of integrated statistical analysis software such as R with an XML database system would bring an XML database closer on par with modern relational databases.

5.5 Conclusion

The use of ASCII logs and relational databases is the status quo. However, as more technologies/standards, businesses, and governments rely on XML for data organization and transport, transitioning from the ASCII and relational database logging/audit systems to an XML based system will likely be warranted.

Appendix A. Databases

A.1 Relational Database

A.1.1 Database Schema. The Following are a series of SQL create table commands used to create the number and types of columns for the each respective table.

A.1.1.1 entry.

```
CREATE TABLE entry
(
  "timestamp" timestamp without time zone,
  entry_tag bigint,
  entry_key character(20) NOT NULL
  -- CONSTRAINT "entry_key_PK" PRIMARY KEY (entry_key)
)
WITH (
  OIDS=FALSE
);
```

A.1.1.2 syscall.

```
CREATE TABLE syscall
(
  entry_key character(20),
  arch character(6),
  syscall character(15),
  personality character(12),
  syscall_sucess boolean,
  exit bigint,
  ppid bigint,
  pid bigint,
```

```
  audit_uid character(12),
  uid character(12),
  gid character(12),
  euid character(12),
  suid character(12),
  fs_uid character(12),
  egid character(12),
  sgid character(12),
  fs_gid character(12),
  terminal character(15),
  com_name character varying,
  exe_name character varying,
  filter_key character varying
  -- CONSTRAINT entry_key_ref FOREIGN KEY (entry_key)
      -- REFERENCES entry (entry_key) MATCH SIMPLE
      -- ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
  OIDS=FALSE
```

A.1.1.3 args.

```
CREATE TABLE args
(
  entry_key character(20),
  argnum bigint,
  argvalue character varying
  -- CONSTRAINT entry_key_ref FOREIGN KEY (entry_key)
      -- REFERENCES entry (entry_key) MATCH SIMPLE
      -- ON UPDATE NO ACTION ON DELETE NO ACTION
```

```
)
WITH (
  OIDS=FALSE
);
```

A.1.1.4 path.

```
CREATE TABLE path
(
  entry_key character(20),
  itemnum bigint,
  "name" character varying(1024),
  inode bigint,
  dev_maj_min character(5),
  permission_mode_oct character varying(15),
  obj_uid character(12),
  raw_dev_maj_min character(5),
  obj_sec_label character varying(15),
  obj_sec_id bigint
  -- CONSTRAINT entry_key_ref FOREIGN KEY (entry_key)
      -- REFERENCES entry (entry_key) MATCH SIMPLE
      -- ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
  OIDS=FALSE
);
```

A.1.1.5 CWD.

```
CREATE TABLE CWD
(
```

```
  entry_key character(20),
  cwd character varying(255)
  -- CONSTRAINT entry_key_ref FOREIGN KEY (entry_key)
      -- REFERENCES entry (entry_key) MATCH SIMPLE
      -- ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
  OIDS=FALSE
);
```

A.1.1.6 credent_acquire.

```
CREATE TABLE credent_acquire
(
  entry_key character(20),
  pid bigint,
  uid character(12),
  audit_uid character(12),
  subj character varying,
  ssid bigint,
  exe_name character varying,
  hostname character varying(32),
  addr_ip character varying(15),
  terminal character(15),
  res_succ_fail character(7),
  accountname character varying(1024)
  -- CONSTRAINT entry_key_ref FOREIGN KEY (entry_key)
      -- REFERENCES entry (entry_key) MATCH SIMPLE
      -- ON UPDATE NO ACTION ON DELETE NO ACTION
)
```

```
WITH (
  OIDS=FALSE
);
```

A.1.1.7 credent_dispose.

```
CREATE TABLE credent_dispose
(
  entry_key character(20),
  pid bigint,
  uid character(12),
  audit_uid character(12),
  subj character varying,
  ssid bigint,
  exe_name character varying,
  hostname character varying(32),
  addr_ip character varying(15),
  terminal character(15),
  res_succ_fail character(7),
  accountname character varying(1024)
  -- CONSTRAINT entry_key_ref FOREIGN KEY (entry_key)
      -- REFERENCES entry (entry_key) MATCH SIMPLE
      -- ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
  OIDS=FALSE
);
```

A.1.1.8 credent_refresh.

```
CREATE TABLE credent_refresh
```

```
(
  entry_key character(20),
  pid bigint,
  uid character(12),
  audit_uid character(12),
  subj character varying,
  ssid bigint,
  exe_name character varying,
  hostname character varying(32),
  addr_ip character varying(15),
  terminal character(15),
  res_succ_fail character(7),
  accountname character varying(1024)
  -- CONSTRAINT entry_key_ref FOREIGN KEY (entry_key)
      -- REFERENCES entry (entry_key) MATCH SIMPLE
      -- ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
  OIDS=FALSE
);
```

A.1.1.9 user_auth.

```
CREATE TABLE user_auth
(
  entry_key character(20),
  pid bigint,
  uid character(12),
  audit_uid character(12),
  subj character varying,
```

```
  ssid bigint,
  exe_name character varying,
  hostname character varying(32),
  addr_ip character varying(15),
  terminal character(15),
  res_succ_fail character(7),
  accountname character varying(1024)
  -- CONSTRAINT entry_key_ref FOREIGN KEY (entry_key)
      -- REFERENCES entry (entry_key) MATCH SIMPLE
      -- ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
  OIDS=FALSE
);
```

A.1.1.10 user_acct.

```
CREATE TABLE user_acct
(
  entry_key character(20),
  pid bigint,
  uid character(12),
  audit_uid character(12),
  subj character varying,
  ssid bigint,
  exe_name character varying,
  hostname character varying(32),
  addr_ip character varying(15),
  terminal character(15),
  res_succ_fail character(7),
```

```
  accountname character varying(1024)
  -- CONSTRAINT entry_key_ref FOREIGN KEY (entry_key)
      -- REFERENCES entry (entry_key) MATCH SIMPLE
      -- ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
  OIDS=FALSE
);
```

A.1.1.11 user_end.

```
CREATE TABLE user_end
(
  entry_key character(20),
  pid bigint,
  uid character(12),
  audit_uid character(12),
  subj character varying,
  ssid bigint,
  exe_name character varying,
  hostname character varying(32),
  addr_ip character varying(15),
  terminal character(15),
  res_succ_fail character(7),
  accountname character varying(1024)
  -- CONSTRAINT entry_key_ref FOREIGN KEY (entry_key)
      -- REFERENCES entry (entry_key) MATCH SIMPLE
      -- ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
```

```
  OIDS=FALSE
);
```

A.1.1.12 user_err.

```
CREATE TABLE user_err
(
  entry_key character(20),
  pid bigint,
  uid character(12),
  audit_uid character(12),
  subj character varying,
  ssid bigint,
  exe_name character varying,
  hostname character varying(32),
  addr_ip character varying(15),
  terminal character(15),
  res_succ_fail character(7),
  accountname character varying(1024)
  -- CONSTRAINT entry_key_ref FOREIGN KEY (entry_key)
      -- REFERENCES entry (entry_key) MATCH SIMPLE
      -- ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
  OIDS=FALSE
);
```

A.1.1.13 user_login.

```
CREATE TABLE user_login
(
```

```
  entry_key character(20),
  pid bigint,
  uid character(12),
  audit_uid character(12),
  subj character varying,
  ssid bigint,
  exe_name character varying,
  hostname character varying(32),
  addr_ip character varying(15),
  terminal character(15),
  res_succ_fail character(7),
  accountname character varying(1024)
  -- CONSTRAINT entry_key_ref FOREIGN KEY (entry_key)
      -- REFERENCES entry (entry_key) MATCH SIMPLE
      -- ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
  OIDS=FALSE
);
```

A.1.1.14 user_start.

```
CREATE TABLE user_start
(
  entry_key character(20),
  pid bigint,
  uid character(12),
  audit_uid character(12),
  subj character varying,
  ssid bigint,
```

```
  exe_name character varying,
  hostname character varying(32),
  addr_ip character varying(15),
  terminal character(15),
  res_succ_fail character(7),
  accountname character varying(1024)
  -- CONSTRAINT entry_key_ref FOREIGN KEY (entry_key)
      -- REFERENCES entry (entry_key) MATCH SIMPLE
      -- ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
  OIDS=FALSE
);
```

A.1.1.15 Indexes.

```
CREATE INDEX args_entry_key on args (entry_key);
CREATE INDEX args_argnum on args (argnum);
CREATE INDEX args_argvalue on args (argvalue);

CREATE INDEX credent_acquire_res_succ_fail on credent_acquire (res_succ_fail);
CREATE INDEX credent_acquire_entry_key on credent_acquire (entry_key);
CREATE INDEX credent_acquire_addr_ip on credent_acquire (addr_ip);
CREATE INDEX credent_acquire_hostname on credent_acquire (hostname);
CREATE INDEX credent_acquire_accountname on credent_acquire (entry_key);
CREATE INDEX credent_acquire_uid on credent_acquire (uid);
CREATE INDEX credent_acquire_pid on credent_acquire (pid);
CREATE INDEX credent_acquire_audit_uid on credent_acquire (audit_uid);
CREATE INDEX credent_acquire_exe_name on credent_acquire (exe_name);
```

```
CREATE INDEX credent_dispose_entry_key on credent_dispose (entry_key);
CREATE INDEX credent_dispose_res_succ_fail on credent_dispose (res_succ_fail);
CREATE INDEX credent_dispose_addr_ip on credent_dispose (addr_ip);
CREATE INDEX credent_dispose_hostname on credent_dispose (hostname);
CREATE INDEX credent_dispose_accountname on credent_dispose (accountname);
CREATE INDEX credent_dispose_uid on credent_dispose (uid);
CREATE INDEX credent_dispose_pid on credent_dispose (pid);
CREATE INDEX credent_dispose_audit_uid on credent_dispose (audit_uid);
CREATE INDEX credent_dispose_exe_name on credent_dispose (exe_name);


CREATE INDEX CWD_entry_key on CWD (entry_key);

CREATE INDEX generic_entries_entry_key on entry (entry_key);
CREATE INDEX generic_entries_timestamp on entry (timestamp);



CREATE INDEX login_entry_key on login (entry_key);
CREATE INDEX login_addr_ip on login (addr_ip);
CREATE INDEX login_hostname on login (hostname);
CREATE INDEX login_accountname on login (entry_key);
CREATE INDEX login_uid on login (uid);
CREATE INDEX login_pid on login (pid);
CREATE INDEX login_audit_uid on login (audit_uid);
CREATE INDEX login_exe_name on login (exe_name);
```

```
CREATE INDEX path_entry_key on path (entry_key);
CREATE INDEX path_itemnum on path (itemnum);
CREATE INDEX path_name on path (name);
CREATE INDEX path_inode on path (inode);
CREATE INDEX path_permission_mode_oct on path (permission_mode_oct);


CREATE INDEX syscall_entry_key on syscall (entry_key);
CREATE INDEX syscall_syscall on syscall (syscall);
CREATE INDEX syscall_exit on syscall (exit);
CREATE INDEX syscall_ppid on syscall (ppid);
CREATE INDEX syscall_pid on syscall (pid);
CREATE INDEX syscall_audit_uid on syscall (audit_uid);
CREATE INDEX syscall_uid on syscall (uid);
CREATE INDEX syscall_gid on syscall (gid);
CREATE INDEX syscall_com_name on syscall (com_name);
CREATE INDEX syscall_exe_name on syscall (exe_name);
CREATE INDEX syscall_filter_key on syscall (filter_key);
--CREATE INDEX syscall_entry_key on syscall (entry_key);
--CREATE INDEX syscall_entry_key on syscall (entry_key);
--CREATE INDEX syscall_entry_key on syscall (entry_key);



CREATE INDEX user_acct_entry_key on user_acct (entry_key);
```

```
CREATE INDEX user_acct_res_succ_fail on user_acct (res_succ_fail);
CREATE INDEX user_acct_addr_ip on user_acct (addr_ip);
CREATE INDEX user_acct_hostname on user_acct (hostname);
CREATE INDEX user_acct_accountname on user_acct (accountname);
CREATE INDEX user_acct_uid on user_acct (uid);
CREATE INDEX user_acct_pid on user_acct (pid);
CREATE INDEX user_acct_audit_uid on user_acct (audit_uid);
CREATE INDEX user_acct_exe_name on user_acct (exe_name);


CREATE INDEX user_auth_entry_key on user_auth (entry_key);
CREATE INDEX user_auth_res_succ_fail on user_auth (res_succ_fail);
CREATE INDEX user_auth_addr_ip on user_auth (addr_ip);
CREATE INDEX user_auth_hostname on user_auth (hostname);
CREATE INDEX user_auth_accountname on user_auth (accountname);
CREATE INDEX user_auth_uid on user_auth (uid);
CREATE INDEX user_auth_pid on user_auth (pid);
CREATE INDEX user_auth_audit_uid on user_auth (audit_uid);
CREATE INDEX user_auth_exe_name on user_auth (exe_name);

CREATE INDEX user_end_entry_key on user_end (entry_key);
CREATE INDEX user_end_res_succ_fail on user_end (res_succ_fail);
CREATE INDEX user_end_addr_ip on user_end (addr_ip);
CREATE INDEX user_end_hostname on user_end (hostname);
CREATE INDEX user_end_accountname on user_end (accountname);
CREATE INDEX user_end_uid on user_end (uid);
CREATE INDEX user_end_pid on user_end (pid);
CREATE INDEX user_end_audit_uid on user_end (audit_uid);
CREATE INDEX user_end_exe_name on user_end (exe_name);
```

```
CREATE INDEX user_err_entry_key on user_err (entry_key);
CREATE INDEX user_err_res_succ_fail on user_err (res_succ_fail);
CREATE INDEX user_err_addr_ip on user_err (addr_ip);
CREATE INDEX user_err_hostname on user_err (hostname);
CREATE INDEX user_err_accountname on user_err (accountname);
CREATE INDEX user_err_uid on user_err (uid);
CREATE INDEX user_err_pid on user_err (pid);
CREATE INDEX user_err_audit_uid on user_err (audit_uid);
CREATE INDEX user_err_exe_name on user_err (exe_name);


CREATE INDEX user_login_entry_key on user_login (entry_key);
CREATE INDEX user_login_res_succ_fail on user_login (res_succ_fail);
CREATE INDEX user_login_addr_ip on user_login (addr_ip);
CREATE INDEX user_login_hostname on user_login (hostname);
CREATE INDEX user_login_accountname on user_login (accountname);
CREATE INDEX user_login_uid on user_login (uid);
CREATE INDEX user_login_pid on user_login (pid);
CREATE INDEX user_login_audit_uid on user_login (audit_uid);
CREATE INDEX user_login_exe_name on user_login (exe_name);


CREATE INDEX user_start_entry_key on user_start (entry_key);
CREATE INDEX user_start_res_succ_fail on user_start (res_succ_fail);
CREATE INDEX user_start_addr_ip on user_start (addr_ip);
CREATE INDEX user_start_hostname on user_start (hostname);
CREATE INDEX user_start_accountname on user_start (accountname);
```

```
CREATE INDEX user_start_uid on user_start (uid);
CREATE INDEX user_start_pid on user_start (pid);
CREATE INDEX user_start_audit_uid on user_start (audit_uid);
CREATE INDEX user_start_exe_name on user_start (exe_name);
```

Bibliography

CC08. C. Coopmans and YangQuan Chen. A general-purpose low-cost compact spatial-temporal data logger and its applications. pages 64–68, Sept. 2008.

CCC04. K.O. Chow, A.Y.K. Chan, and K.S. Cheung. An xml approach to student usage reflection in online courses. pages 339–344, Sept. 2004.

CEE08. Common event expression white paper, Jun 2008. URL `http://cee.mitre.org/documents.html`.

Cor03. Jason Cornpropst. Canonical situation data format: The common base event v1.0.1, 2003. URL `http://www.eclipse.org/tptp/platform/documents`.

Cor08. MITRE Corporation. Security content automation protocol, 2008. URL `http://nvd.nist.gov/scap/docs/SCAP.doc`.

EXI09. Efficient xml interchange evaluation. Technical report, World Wide Web Consortium, Apr 2009. URL `http://www.w3.org/TR/exi-evaluation`.

GPR+03. M.A. Goncalves, G. Panchanathan, U. Ravindranathan, A. Krowne, E.A. Fox, F. Jagodzinski, and L. Cassel. The xml log standard for digital libraries: analysis, evolution, and deployment. pages 312–314, May 2003.

Gra93. Jim Gray. *The Benchmark Handbook for Database and Transaction Systems*. Morgan Kaufmann series in data management systems. M. Kaufmann Publishers, San Mateo, CA, 2nd edition, 1993.

HB99. Philip M. Hallam-Baker. Extended log file format, 1999. URL `http://www.w3.org/TR/WD-logfile.html`.

IBM03. IBM. Common base event for logging documentation version 1.0, 2003. URL `http://www.eclipse.org/tptp/platform/documents/resources/cbe.pdf`.

IBM08. IBM. Enterprise multiplatform auditing, 2008. URL `http://www.redbooks.ibm.com/redbooks/pdfs/sg247472.pdf`.

IDM04. Intrusion detection message exchange format. web, 2004. URL `http://xml.coverpages.org/idmef.html`.

Kro02. David M. Kroenke. *Database Processing Fudamentals, Design & Implementation*. Prentice Hall, Upper Saddle River, NJ, 07458, 9th edition, 2002.

LOG01. Logml 1.0 draft specification, 2001. URL `http://www.cs.rpi.edu/puninj/LOGML/draft-logml.html`.

PKZ02. JR Punin, MS Krishnamoorthy, and MJ Zaki. LOGML: Log markup language for Web usage mining. In Kohavi, R and Masand, BM and Spiliopoulou, M and Srivastava, J, editor, *WEBKDD 2001 - Mining Web Log Data Across All Customers Touch Points*, volume 2356 of *Lecture Notes In Artificial Intelligence*, pages 88–112. Springer-Verlang Berlin, Heidelberger Platz 3, D-14197 Berlin, Germany, 2002.

RXB09. Huw Read, Konstantinos Xynos, and Andrew Blyth. Presenting devise: Data exchange for visualizing security events. *Computer Graphics and Applications, IEEE*, 29(3):6–11, May-June 2009.

Sch07. Andreas Schuster. Introducing the Microsoft Vista event log file format. *DIGITAL INVESTIGATION*, 4(Suppl. 1):S65–S72, SEP 2007.

SDE. The security device event exchange. URL `http://www.icsalabs.com/icsa/topic.php?tid=b2b4$52d6a7ef-1ea5803f$4c69-ff36f9b5`.

Wal07. Pricilla Walmsley. *XQuery*. O'Reilly, 2007.

WG05. XML Binary Characterization Working Group. Xml binary characterization use cases. Technical report, World Wide Web Consortium, Mar 2005. URL `http://www.w3.org/ TR/xbc-use-cases`.

Wor03. Binary Interchange Workshop. Report from the w3c workshop on binary interchange of xml information item sets. Technical report, World Wide Web Consortium, Sep 2003. URL `http://www.w3.org/2003/08/binary-interchange-workshop/Report.html`.

Wor08. World Wide Web Consortium. *Efficient XML Interchange (EXI) Format 1.0*, Sept 2008. URL `http://www.w3.org/ TR/exi`. Draft.

XLF01. Extensible log format initiative. web-based, 2001. URL `http://xml.coverpages.org/ xlf.html`.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
10–09–2010	Master's Thesis	Sept 2008 — Sept 2010

4. TITLE AND SUBTITLE

A
Comparative Analysis of
ASCII and XML Logging Systems

5a. CONTRACT NUMBER

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Eric Hanington, Civ, USAF

5d. PROJECT NUMBER
N/A

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/ENG)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCO/ENG/10-17

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Security Agency (LtCol Joseph Wolfkiel)
9800 Savage Rd, Suite 6767
Fort Meade, MD 20755-6767
(410-854-5401 and j.wolfki@radium.ncsc.mil)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approval for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This research compares XML and ASCII based event logging systems in terms of their storage and processing efficiency. XML has been an emerging technology, even for security. Therefore, it is researched as a logging system with the mitigation of its verbosity. Each system consists of source content, the network transmission, database storage, and querying which are all studied as individual parts. The ASCII logging system consists of the text file as source, FTP as transport, and a relational database system for storage and querying. The XML system has the XML files and XML files in binary form using Efficient XML Interchange encoding, FTP as transport using both XML and binary XML, and an XML database for storage and querying. Further comparisons are made between the XML itself and binary XML, as well as binary XML to ASCII text when comparing file sizes and transmission efficiency. XML itself is a poor choice for hard drive and network transport time compared to ASCII. However, in a binary form, it uses less hard drive space and network resources. Because no XML databases support a binary XML, it is loaded without any optimization. The ASCII loads into the relational database with less time than XML into its database. However, querying each database, neither outperforms the other as one query results in shorter time for one, and another query results in a shorter time for the other. Therefore, XML and/or its binary form, is a viable candidate for use as a comprehensive logging system.

15. SUBJECT TERMS

XML, Logs, XML Logs,EXI, Efficient XML Interchange, XML Schema, XQuery

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Rusty Baldwin
U	U	U	UU	85	19b. TELEPHONE NUMBER *(include area code)* (937) 255–6565, ext 4445; rusty.baldwin@afit.edu